Alma Mater Studiorum · Università di Bologna

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

# Progettazione e implementazione di applicazioni context-aware su dispositivi mobili

**Tesi di Laurea in Informatica**

**Relatore:**
**Chiar.mo Prof.**
**Luciano Bononi**

**Presentata da:**
**Marco Di Nicola**

**Correlatore:**
**Dott.**
**Marco Di Felice**

**Sessione II**
**2011/2012**

# Abstract

Il progetto descritto in questo documento consiste fondamentalmente nell'integrazione di applicazioni context-aware su dispositivi mobili con reti di sensori e nello studio delle problematiche derivanti, vantaggi e potenziali utilizzi.
La rete è stata costruita sfruttando l'insieme di protocolli per comunicazioni via radio Zigbee, particolarmente adatti per interazione tra dispositivi a basso consumo energetico e che necessitano di uno scarso tasso di trasferimento di dati.
Le informazioni ottenute da sensori di varia natura sono processate da microcontrollori Arduino, scelti per la loro versatilità di utilizzo e design open source.
Uno o più dispositivi sono designati per aggregare i dati rilevati dai singoli nodi in un unico pacchetto di informazioni, semanticamente correlate tra loro, quindi emetterle in broadcast su una diversa interfaccia di rete, in modo che diverse applicazioni esterne in ascolto possano riceverle e manipolarle.
È stato utilizzato un protocollo specifico per la comunicazione tra i microcontrollori e le applicazioni che si interfacciano con la rete, costruito su misura per dispositivi con risorse limitate.
L'applicazione context-aware che interagisce con la rete è stata sviluppata su piattaforma Android, la cui particolare flessibilità favorisce una migliore capacità di gestire i dati ottenuti.
Questa applicazione è in grado di comunicare con la rete, manipolare i dati ricevuti ed eventualmente intraprendere azioni specifiche in totale indipendenza dal suo utilizzatore.
Obiettivo del progetto è quello di costruire un meccanismo di interazione tra le tecnologie più adattivo e funzionale possibile.

# Indice

# Introduzione

Due fra i paradigmi più promettenti introdotti dalla Computer Science negli ultimi anni sono quelli di Internet of Things e Context Aware Computing.

Il primo si riferisce all'integrazione di qualunque oggetto fisico nella rete Internet, fornendone una rappresentazione virtuale capace di identificarlo e trasmetterne informazioni. Le potenzialità offerte da questa rete sono pressoché illimitate: un incremento e ottimizzazione dei processi produttivi, automatizzazione di edifici, commercio intelligente e migliorie nei trattamenti sanitari. Gli oggetti inclusi in questa rete di informazioni comprendono anche quelli di uso comune: vestiti, chiavi, contenitori di medicinali; tutti identificati univocamente attraverso chip RFID (Radio Frequency Identifier) o codici a barre.

Il concetto di Context Aware Computing descrive lo sviluppo di tecnologie e applicazioni capaci di rilevare dati dal contesto circostante e reagire di conseguenza con determinate azioni. Sebbene l'idea si adatti a qualunque tipo di piattaforma, un applicativo in esecuzione su dispositivi mobili offre sicuramente maggiori potenzialità. Si pensi ad una applicazione in esecuzione su smartphone che effettui una stima sulla velocità di movimento del suo possessore, sfruttando sensori quali accelerometro e giroscopio, determini il mezzo utilizzato per spostarsi e reagisca lanciando determinate applicazioni: un semplice esempio potrebbe essere la riproduzione di un brano musicale, se dovesse stabilire che l'utente stia facendo jogging.

Il progetto descritto in questo documento comporta l'integrazione dei due concetti: la creazione di una rete di dispositivi, comprensivi di attuatori e sensori capaci di rilevare dati ambientali quali temperatura, luminosità, umidità o relativi a oggetti di ogni genere, e la sua interazione con un'applicazione Context-Aware in esecuzione su un dispositivo smartphone.

I dispositivi utilizzati per formare la rete consistono in microcontrollori Arduino, scelti per il loro design aperto e flessibilità di utilizzo.
Per instaurare una connessione fra loro vengono utilizzati i protocolli di comunicazione wireless ZigBee, ideali per reti con scarso data rate e dispositivi a basso consumo energetico, per i quali si richiede quindi un'alimentazione a batteria con la maggiore autonomia possibile.
La rete così costituita assume una topologia a stella, delineando due ruoli fondamentali fra i dispositivi:

- Un singolo nodo denominato Sink, dotato di una doppia interfaccia di rete: quella ZigBee, con la quale riceve informazioni dagli altri microcontrollori e quella Ethernet, attraverso la quale comunica con i dispositivi mobili esterni.

  Il Sink è provvisto di alimentazione elettrica e ha il compito fondamentale di raccogliere e aggregare informazioni, per poi spedirle in broadcast, periodicamente, sulla sua interfaccia Ethernet.

- Un numero variabile di End Device, dispositivi forniti di attuatori e sensori da cui effettuare rilevazioni, che trasmettono periodicamente i dati rilevati al Sink. Per questi dispositivi è prevista un'alimentazione a batteria.

È importante osservare che, data la natura della comunicazione via broadcast, più applicazioni di natura eterogenea potrebbero ricevere ed elaborare i dati in simultanea.

Dall'altro lato della comunicazione si ha un'applicazione Context Aware in esecuzione su dispositivo mobile.
Questa applicazione è in grado di ricevere e decodificare le informazioni emesse dal Sink, fornendo all'utente un'interfaccia su esse e processandole in base a determinati criteri: potrebbe essere richiesto che la media aritmetica dei valori di temperatura rilevati in un edificio non superi una soglia massima, o che un certo numero di switch siano disattivati dopo un orario prestabilito.
L'applicazione può in sostanza compiere delle scelte in base ai dati ottenuti dal Sink, eseguendo delle azioni in maniera del tutto autonoma e indipendente dall'utilizzatore.
La piattaforma scelta per lo sviluppo di questa applicazione è quella Android, data la sua notevole diffusione e flessibilità nell'utilizzo.

Viene utilizzato (e descritto in questo documento) un protocollo di comunicazione molto conciso, al fine di minimizzare la quantità di dati trasferiti sulla rete ZigBee e ottimizzare l'uso della scarsa quantità di memoria di cui i microcontrollori sono dotati. L'applicazione mobile è dotata di mezzi per tradurre questo protocollo in diversi formati, leggibili da essere umano e interscambiabili tra loro, per esempio in base alla lingua dell'utilizzatore.

Nel seguito di questo documento viene innanzitutto fornita una descrizione delle tecnologie utilizzate: lo standard wireless ZigBee, la piattaforma Arduino e il sistema operativo Android.
Successivamente sarà esaminata in maggiore dettaglio la costruzione della rete e l'implementazione dell'applicazione, ponendo enfasi sulle scelte progettuali effettuate e problematiche incontrate.
Infine sarà fornita una breve panoramica sulle potenzialità di utilizzo che l'uso di queste tecnologie offre, illustrando alcuni possibili casi reali.

# Parte I

# Stato dell'Arte

# Capitolo 1

# ZigBee

## 1.1 802.15.4

L'esigenza di un data-rete sufficientemente elevato da consentire la connettività di periferiche altamente interattive (come quelle usate dai Personal Computer), ma al contempo riducibile ai livelli tipici richiesti da sensori ed applicazioni orientate al controllo e automazione delle infrastrutture, ha portato alla nascita di una tipologia di rete denominata **LR-WPAN** (Low Rate Wireless Personal Area Network).
Una LR-WPAN è una rete di comunicazione semplice e a basso costo, orientata verso applicazioni a basso consumo e throughput, ossia quantità di informazione trasmessa, non elevato.

Lo standard **IEEE 802.15.4** nasce per far fronte a questa esigenza e viene ufficialmente approvato nell'estate del 2003, definendo un protocollo di trasmissione a basso livello tramite comunicazione radio, tra diversi dispositivi volti a formare una LR-WPAN.
Lo standard definisce, più in particolare, le specifiche del livello fisico (PHY) e medium access control (MAC) al fine di garantire una modalità di connessione wireless a basso data-rate tra dispositivi fissi, portatili o mobili che necessitano di un basso consumo di potenza, ovvero lunga durata delle batterie a bordo e che tipicamente lavorano in uno spazio operativo (POS: Personal Operating Space) dell'ordine di qualche decina di metri.

Le caratteristiche fondamentali di questa tipologia di rete sono:

Data-rate di 250 kbit/s, 40 kbit/s e 20 kbit/s.

Operabilità in configurazione a stella o peer-to-peer.

Indirizzi a 16 o 64 bit.

Accesso al canale in modalità CSMA-CA.

Basso consumo di energia.

Indicazione della qualità del canale.

16 canali nella banda attorno a 2.4 GHz, 10 canali nella banda attorno a 915 MHz, un canale a 868 MHz.

**ZigBee, WiFi™, and Bluetooth™ compared**

| NAME | ZIGBEE | WiFi | BLUETOOTH |
|---|---|---|---|
| Standard | 802.15.4 | 802.11a,b,g | 802.15.1 |
| Application | Monitoring and control | Web, e-mail, video | Cable replacement |
| System resources | 50 to 60 Kbytes | > 1 Mbyte | > 250 Kbytes |
| Battery life (days) | 100 to > 1000 | 1 to 5 | 1 to 7 |
| Network size | 65, 536 | 32 | 7 |
| Bandwidth (Kb/s) | 20 to 250 | 11,000 | 720 |
| Maximum transmission range (m) | 100+ | 100 | 10 |
| Success metrics | Reliability, power, cost | Speed, flexibility | Cost, convenience |

Figura 1.1: Comparazione tra i diversi standard Wireless

Le differenze fra le specifiche dello standard 802.15.4 e quelle di 802.11 (wifi) e 802.15.1 (bluetooth), tra le quali spiccano le risorse consumate e l'autonomia garantita per i singoli nodi, evidenziano come lo standard 802.15.4 sia quindi particolarmente adatto a vaste reti di piccoli dispositivi che necessitano di lunghi periodi di autonomia.

### 1.1.1 Livello Fisico (PHY)

Il livello più basso definito dallo standard 802.15.4 è quello fisico (PHY), il quale fornisce due differenti servizi: quello dati e quello gestione.

Le caratteristiche principali implementate a questo livello sono l'attivazione e la disattivazione del trasmettitore radio (secondo diverse politiche di sleep dei dispositivi), il rilevamento dell'energia (ED), l'indicazione della qualità del collegamento (LQI), la selezione del canale, la stima della disponibilità del canale (CCA) e la trasmissione e ricezione dei pacchetti sul mezzo fisico (canale radio).
La velocità di trasmissione è di 250 kbit/s a 2.4 GHz, 40 kbit/s a 915 MHz e 20 kbit/s a 868 MHz.
L'indicazione della qualità del collegamento e il controllo sulla disponibilità del canale costituiscono due delle caratteristiche più importanti di questo livello, utilizzate in particolar modo da quelli superiori per costruire una rete stabile ed efficiente.

L'ED rappresenta una stima del segnale ricevuto, codificata in una variabile intera di dimensione 8 bit (da 0x00 a 0xFF), e viene sfruttata per valutare la qualità di un determinato canale, al fine di selezionare il migliore fra quelli disponibili.

La stima di disponibilità del canale (CAA) è altrettanto importante e viene utilizzata dall'algoritmo CSMA-CA per evitare collisioni nel trasferimento di dati ad un medesimo nodo, accertandosi che il canale non sia già in uso da un altro dispositivo, prima di avviare una trasmissione.

### 1.1.2 Livello Medium Access Control (MAC)

Il livello MAC è responsabile fondamentalmente di coordinare l'accesso al canale, introdurre un meccanismo di indirizzamento dei nodi, permettere il rilevamento (Discovery) da parte di un dispositivo, generare e controllare frames di bit, implementare meccanismi di acknowledgement e verifica del numero di sequenza dei frames.

Lo standard 802.15.4 definisce 4 strutture di frame:

**Beacon** frame: trasmessi dal Coordinator (si veda 1.2.1) per trasmettere beacons in broadcast. Utilizzati per implementare particolari soluzioni per accesso coordinato al canale e/o favorire il join di altri nodi alla rete.

| **Octets: 2** | **1** | **4/10** | **2** | **variable** | **variable** | **variable** | **2** |
|---|---|---|---|---|---|---|---|
| Frame control | Sequence number | Addressing fields | Superframe specification | GTS fields | Pending address fields | Beacon payload | FCS |
| MHR | | | MAC payload | | | | MFR |

Figura 1.2: Formato di un Beacon frame

**Data** frame: utilizzati da tutti i nodi per trasmettere dati.

| **Octets: 2** | **1** | **(see 7.2.2.2.1)** | **variable** | **2** |
|---|---|---|---|---|
| Frame control | Sequence number | Addressing fields | Data payload | FCS |
| MHR | | | MAC payload | MFR |

Figura 1.3: Formato di un Data frame

**Acknowledgment** frame: utilizzati per confermare la ricezione di un frame.

| Octets: 2 | 1 | 2 |
|---|---|---|
| Frame control | Sequence number | FCS |
| MHR | | MFR |

Figura 1.4: Formato di un Acknowledgment frame

**MAC Command** frame: utilizzati per specificare azioni di controllo sulla rete quali richieste di associazione o abbandono.

| Octets: 2 | 1 | (see 7.2.2.4.1) | 1 | variable | 2 |
|---|---|---|---|---|---|
| Frame control | Sequence number | Addressing fields | Command frame identifier | Command payload | FCS |
| MHR | | | MAC payload | | MFR |

Figura 1.5: Formato di un MAC Command frame

Poiché il mezzo di comunicazione (il canale radio) è condiviso fra tutti i dispositivi, occorre disporre di qualche metodo di arbitraggio della trasmissione, per evitare che due o più nodi inviano pacchetti contemporaneamente.
Le due alternative possibili sono i beacons, frames emessi periodicamente dal Coordinator per effettuare uno scheduling delle trasmissioni (meccanismo adatto a reti in cui i dispositivi devono trasmettere molto frequentemente) o il **CSMA-CA** (Carrier Sense Multiple Access with Collision Avoidance).
Quest'ultimo algoritmo prevede che, prima di trasmettere, ogni nodo si ponga in ascolto sul canale: se questo viene rilevato come occupato il nodo ritenterà successivamente, con un ritardo casuale.
Il CSMA-CA è più adatto a reti i cui nodi sono inattivi per la maggior parte del tempo, quindi una rete di sensori nella quale le rilevazioni non debbano essere effettuate con intervalli di tempo troppo brevi.

## 1.2 Livelli superiori: lo standard ZigBee

La specifica ZigBee consiste in uno stack di protocolli posti al di sopra di quelli PHY e MAC dello standard 802.15.4.
Questa specifica è stata sviluppata da un consorzio di compagnie, che conta più di 300 membri ed è conosciuto con il nome di ZigBee Alliance.
I livelli aggiunti dallo standard ZigBee sono i seguenti:

- Un livello **Network** che consente la vera e propria creazione di reti (associando o dissociando uno o più dispositivi ad un nodo Coordinator), aggiunge la possibilità di instradare pacchetti di informazioni attraverso un numero arbitrario di dispositivi (creando quindi reti multihop) e implementa politiche di sicurezza.
- Un livello **Applicativo** che definisce una serie di funzionalità aggiuntive, quali discovery di nodi nella rete in base a determinati servizi offerti e meccanismi avanzati di crittografia. Questo livello comprende una serie di profili utili a definire un'interfaccia verso alcune tipologie di applicazioni esterne che operano tramite sensori o attuatori. La trattazione in dettaglio delle funzionalità offerte da questo livello non è compresa nel documento, in quanto saranno utilizzate diverse tecnologie per raggiungere lo stesso scopo.

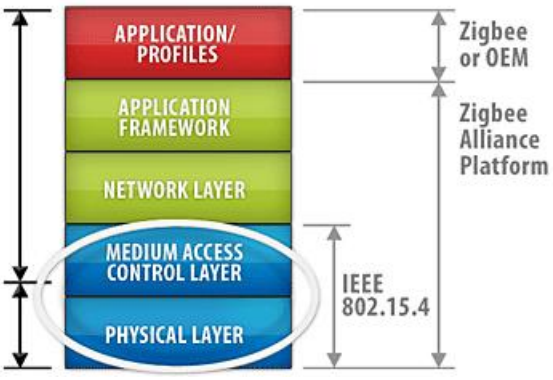



Figura 1.6: Stack di protocolli IEEE 802.15.4 / ZigBee

### 1.2.1 Livello Network

Il livello Network, allo scopo di formare una rete di dispositivi ZigBee interconnessi (quindi una PAN), delinea tre diversi profili che i singoli nodi possono assumere: **Coordinator**, **Router** e **End Device**.

Le principali responsabilità di questo livello consistono nel creare una nuova rete, aggiungervi dispositivi assegnando loro un indirizzo, rimuoverli e instradare pacchetti attraverso molteplici nodi.

#### Coordinator

Unico in una PAN, il Coordinator ha fondamentalmente il compito di crearla, selezionando un opportuno canale e PAN ID a 16 bit e permettere ad altri dispositivi di connettervisi, mantenendone le informazioni.
Il Coordinator può inoltre instradare i pacchetti ricevuti verso altri nodi e bufferizzare quelli destinati ad un certo numero di End Device posti in sleep, i quali prendono la denominazione di figli.

Il primo passo nella creazione di una rete consiste nel definire una lista di canali selezionabili: per fare ciò il Coordinator effettua un controllo sulla qualità di ogni canale (sfruttando le funzionalità offerte dal sottostante livello fisico), definita da una stima sulla potenza del segnale ricevuto. Se il segnale è particolarmente forte il canale è probabilmente utilizzato e viene quindi scartato dalla lista di canali selezionabili.
Una volta ultimato questo procedimento, denominato Energy Scan, il Coordinator trasmette un beacon in broadcast su ognuno dei canali rimasti; a questo risponderanno con un altro beacon tutti i Routers e Coordinators attivi su altre reti, includendo nel frame l'identificativo a 16 bit per la loro PAN.
Quindi il Coordinator seleziona un PAN id casuale fra quelli inutilizzati e crea la rete, secondo determinate politiche di sicurezza (Joining permesso per un tempo limitato o ad un numero limitato di dispositivi).

Quando un altro dispositivo (Router o End Device) emette una richiesta di associazione alla rete, il Coordinator accettandola ne diventa padre: memorizza il network ID ad esso assegnato, nonchè quello a 64 bit di fabbricazione, in una delle voci di una tabella a grandezza fissa che prende nome di **Child table** e si fa carico di bufferizzare eventuali pacchetti destinati a quel dispositivo, mentre questo è in stato di sleep.
Per assolvere questi compiti è necessario che il Coordinator sia alimentato da rete elettrica e non a batteria; inoltre su firmware Xbee ZB 2x6x la dimensione della tabella è limitata a 10 voci, rendendo necessario l'utilizzo di Router per reti con un numero di nodi superiori a 11.

### Router

Il Router opera in modo del tutto simile al Coordinator, salvo il non poter creare direttamente una PAN (la cerca esattamente come farebbe un End Device) e potendo accettare fino a 12 nodi figli (sempre con firmware di tipo Xbee ZB 2x6x).
La ragione principale per cui la ZigBee Alliance ha introdotto nelle sue specifiche questo tipo di dispositivi è quello di permettere la creazione di reti multihop, il cui numero massimo di nodi è limitato dal range di indirizzi assegnabili con 16 bit.

Una volta alimentato un Router effettua periodicamente una PAN scan (o active scan) di tutti i canali disponibili, trasmettendo su di essi un beacon che rappresenta una richiesta di informazioni.
A questo beacon rispondono con un altro tutti i Routers e Coordinators attivi su altre reti, indicando se la PAN accetta nuovi elementi (quindi le child table dei Routers o del Coordinator hanno voci inutilizzate).
Se non sono trovate reti disponibili il Router ricomincia la scansione dei canali, dopo un'attesa relativa al numero di tentativi (9 al minuto nei primi 5 minuti, 3 al minuto successivamente).
Una volta selezionata una rete, il Router richiede l'associazione ad essa al dispositivo che ha risposto al beacon iniziale. Se questo lo accetta (quindi le politiche di sicurezza lo consentono) il Router ottiene un identificativo a 16 bit selezionato casualmente fra quelli disponibili.

### End Device

Un End Device rappresenta un nodo semplice all'interno della rete: non può salvare informazioni relative ad altri nodi (salvo forse l' ID del genitore) né bufferizzare pacchetti. Si comporta esattamente come un Router nel cercare una PAN valida a cui unirsi, ma a differenza di esso può scendere ad un livello molto basso di consumo energetico, entrando in uno stato di sleep.
In questo stato il dispositivo non può ricevere i pacchetti destinati a lui, necessitando quindi che il nodo genitore li bufferizzi in attesa che la fase di sleep termini.

Generalmente sono previste due diverse modalità di sleep:

- La modalità **Pin sleep** consente ad un microcontrollore esterno di determinare i periodi di sleep e di veglia del modulo mediante asserzione di un pin denominato Sleep-RQ. Il segnale è di tipo digitale: un valore HIGH (5 volt) fa in modo che, al termine di eventuali trasferimenti in atto, il modulo si iberni fino a nuova asserzione del medesimo pin.
- La modalità **Cyclic Sleep** prevede che il dispositivo vada in sleep per un lasso di tempo preimpostato (determinato moltiplicando il valore di un contatore per 10 millisecondi), al termine del quale seguirà un breve ciclo di polling sul nodo padre, per verificare che non abbia dati destinati a lui in buffer.

### Dati persistenti

Tutti i nodi conservano le informazioni relative alla PAN formata (Coordinator) o alla quale sono associati (Router e End Device), quali PAN id e canale scelto, su memoria persistente.
Occorre un criterio per stabilire quando, qualora la rete non fosse più raggiungibile (per esempio a causa dello spostamento del nodo), sia necessario lasciarla, iniziando un nuovo active scan.
Le soluzioni attuate differiscono a seconda che il dispositivo sia un Router o un End Device.
Un Router effettua l'interrogazione periodica del Coordinator (sfruttando il suo identificativo a 64 bit, assegnato alla fabbricazione) per verificare che la rete sia ancora attiva (meccanismo di **Network Watchdog**); se questo non risponde entro un limite fissato di tentativi, il Router lascia la rete e ne cerca una nuova.
Diversamente un End Device effettuerà un **Orphan Scan**: il suo identificativo a 64 bit verrà trasmesso in broadcast; i Routers e Coordinators circostanti lo capteranno e faranno un confronto con le voci memorizzate nella loro child table, rispondendo se viene trovato un riscontro. In caso di esito negativo l'End Device lascia la rete e ne cerca una nuova.

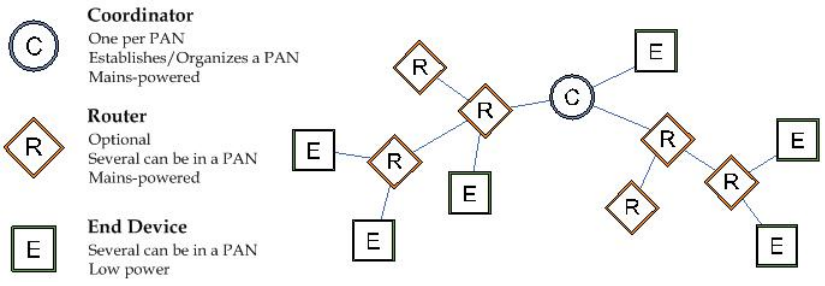



Figura 1.7: Schema di una rete ZigBee

## 1.3 Comunicazione con altri dispositivi

Generalmente i moduli che sfruttano i protocolli ZigBee implementano due diverse modalità di comunicazione con il sistema al quale sono connessi:

- Una modalità detta **Trasparente**, che mette in comunicazione diretta il modulo con il sistema associato. In questo modo il buffer d'ingresso di un modulo si riflette automaticamente in quello di uscita dell'altro e viceversa.

  Sfruttando questa modalità la comunicazione seriale tra due microcontrollori remoti viene notevolmente semplificata, necessitando semplicemente che i due siano integrati con modem connessi alla medesima PAN e configurati per spedirsi pacchetti di informazioni.

  Questo meccanismo impone che, se il microcontrollore avesse necessità di comunicare direttamente con il modem ad esso connesso (qualora dovesse riconfigurarlo, per esempio), sia necessaria una sequenza particolare di caratteri ASCII trasmessi, per entrare in modalità Command (usualmente è la sequenza +++).
  In modalità Command alcuni parametri memorizzati nel firmware del dispositivo, quali un filtro per i canali da controllare o per le PAN da accettare, possono essere letti o scritti usando una serie di comandi chiamati **AT**.

  Una volta abilitata la modalità Command, il dispositivo attende l'immissione di comandi ignorando tutti i dati ricevuti che non soddisfino il formato previsto, lasciando la modalità se non ne riceve alcuno entro un limite di tempo.

- Una modalità più complessa: **API**.

  Questa prevede un formato di pacchetti specifico per instaurare una comunicazione tra microcontrollore e modem, allo scopo di indicare il tipo di operazione richiesta.

  Sebbene sia decisamente più complessa e comporti un notevole overhead sui pacchetti trasferiti, la modalità API presenta una maggiore flessibilità di quella trasparente, consentendo di modificare i parametri del firmware dinamicamente senza perdita di dati.

Il dispositivo che utilizza il modem ha quindi necessità di specificare i campi che compongono il frame da spedire.


Figura 1.8: Struttura di un frame API basilare

Entrando in maggiore dettaglio nelle possibilità aggiuntive che la modalità API offre, possiamo includere anche l'identificazione dell'indirizzo dal quale è arrivato un messaggio, la ricezione di acknowledgment relativi ai frames trasmessi, la lettura del RSSI (Received Signal Strength Indication) sul canale utilizzato e la trasmissione a destinatari diversi, senza riconfigurazione del modulo.
Si noti che quest'ultima caratteristica rappresenta un punto fondamentale nella differenza fra le due modalità: usando quella trasparente i bytes scritti vengono automaticamente incapsulati nel payload di un frame di dati, il cui indirizzo di destinazione è staticamente scritto nel firmware; usando la modalità API è possibile specificare i campi dell'header per il frame da spedire, selezionando dinamicamente diversi destinatari.

Segue un esempio di frame API utilizzato per spedire la stringa "Hi!" in broadcast sulla rete:

```
Frame:

7E 0011 10 01 00000000 0000FFFF FFFE 00 00 48 69 21 AD

Composizione:

7E - Delimitatore di inizio frame

0x0011 - Lunghezza del frame
```

```
0x10 - API ID
(indica il tipo di operazione)

0x01 - Frame ID

0x00000000 0000FFFF - Indirizzo a 64 bit per il broadcast

0xFFFE - Indirizzo di rete a 16 bit per il broadcast

0x00 - Numero di hops nel broadcast del frame
(0 = numero massimo)

0x00 - Opzioni

0x48 69 21 - Rappresentazione ASCII della stringa "Hi!"

0x21 - Checksum
(0xFF meno la somma di tutti i bytes dopo il campo relativo
alla lunghezza)
```

Come accennato in precedenza (ed evidenziato da questo esempio) lo svantaggio critico di questa modalità di trasmissione è proprio l'overhead derivante dall'aggiunta dei campi che formano il frame: 12 bytes scritti per inviare una stringa composta da 3.
Nonostante questo la modalità API costituisce la migliore alternativa per gestire una rete complessa, specialmente se essa fa uso delle funzionalità offerte dal livello Applicativo dei protocolli ZigBee.

# Capitolo 2

# Arduino

Il progetto Arduino nasce presso l'Interaction Design Institute nella città Italiana di Ivrea, nel 2005. Il team che lo sviluppa è composto da Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis.

Il progetto nasce come sviluppo parallelo (fork) della piattaforma **Wiring**, ideata e creata come tesi di Master all'Interaction Design Institute da Hernando Barragá, un allievo di Banzi.

Figura 2.1: Logo ufficiale del progetto Arduino

Lo scopo del progetto è quello di fornire a ingegneri, hobbisti e studenti una piattaforma di prototipazione elettronica basata su hardware e software molto flessibili, semplici da modificare e adattare alle proprie esigenze.
Infatti lo schema del circuito stampato e l'elenco dei componenti necessari, nonché il codice sorgente dell'IDE (Ambiente di sviluppo) e librerie integrate, sono rilasciati rispettivamente sotto licenza Creative Commons e GPLv2 e scaricabili gratuitamente dal web.

Il successo di questo progetto (a Ottobre del 2008 risultavano più di 50'000 esemplari di Arduino venduti in tutto il mondo) deriva fondamentalmente dalla semplicità di utilizzo e dalla significativa riduzione dei costi della piattaforma, rispetto a molti prodotti disponibili sul mercato.
Arduino ha successivamente trovato largo impiego presso piccole e medie aziende, per utilizzi di automazione e controllo di strutture.

Sono state prodotte numerose versioni della scheda, nonché numerosi accessori atti a facilitarne l'utilizzo o favorire la comunicazione del microcontrollore con dispositivi esterni; alcuni di questi prendono il nome di **Shield**, in quanto i loro pins possono essere attaccati direttamente ai connettori esposti della scheda, utilizzandone alcuni e riproducendo gli altri per l'uso da parte di ulteriori dispositivi.

## 2.1 Specifiche tecniche

Sebbene esista una vasta gamma di schede Arduino pre-assemblate e acquistabili online dall'Arduino Store o in negozi specializzati, un generico dispositivo risponde alle seguenti caratteristiche:

- Un chip a 8 bit della serie megaAVR, prodotti dalla Atmel (ATmega8, ATmega168, ATmega328, ...).
- Una memoria non volatile flash, generalmente da 32 KB, contenente il bootloader (una versione base da 0.5 a 2 KB, anch'esso riprogrammabile) e lo **sketch** iniettato nel microcontrollore (il codice eseguito).
- Una quantità che va dai 1024 B agli 8 KB di memoria volatile SRAM (Static Random Access Memory), utilizzata per lo stack dello sketch in esecuzione. Per una mole significativa di dati da processare al programmatore è offerta la possibilità di usare direttamente la memoria flash, tramite la keyword PROGMEM (memoria di programma).
- Una esigua quantità di EEPROM utilizzabile come storage per memorizzare dati persistenti. L'accesso a questa porzione di memoria è facilitato da due diverse librerie: EEPROM ed EEPROMex, per funzionalità estese.
- Una serie di ingressi esposti per pins analogici e digitali, allo scopo di favorirne l'uso da parte dei circuiti di altri componenti (quali sensori e attuatori).
- Un regolatore lineare di tensione a 5 volt e un oscillatore a cristallo a 16 MHz per stabilizzare il segnale di clock.
- Interfaccia seriale EIA RS-232, usata per alimentazione e caricamento di sketch sul dispositivo (convertito in flusso seriale a partire da un ingresso USB o mini-USB)

## 2.2 Programmazione

Il caricamento del codice sul microcontrollore avviene generalmente attraverso l'interfaccia seriale EIA RS-232, secondo una conversione USB-to-Serial.
È possibile caricare il codice di uno sketch direttamente attraverso l'IDE ufficiale: una derivazione di quello utilizzato per il linguaggio di programmazione Processing e integrato nel progetto Wiring, scritto in Java e quindi cross-platform.
L'IDE integra al suo interno una libreria denominata Wiring, come il progetto da cui è partito, scritta in C/C++ e comprensiva di numerose funzioni atte a semplificare le operazioni di input/output sugli ingressi analogici e digitali di I/O.

Un tipico sketch per Arduino, scritto in un linguaggio Object-Oriented e basato sulla sintassi C/C++, presenta una struttura fissa:

```
void setup() {

Operazioni di inizializzazione
.....

}
```

La funzione **setup** viene eseguita al termine del bootloader: prevede tutte le eventuali operazioni iniziali da eseguire una sola volta (inizializzazione di particolari variabili, impostazione di pins analogici o digitali per input o output, ...).

```
void loop() {

Operazioni da eseguire ciclicamente
.....
delay(num_millisecondi);

}
```

La funzione **loop** viene eseguita ripetutamente dopo setup, in un ciclo infinito.
Le istruzioni inserite in questo blocco di codice sono quelle che il microcontrollore dovrà eseguire finché è alimentato.
Potrebbero essere operazioni che comprendono il controllo periodico di sensori, la lettura di dati in ingresso da porta seriale o interfaccia di rete.
La libreria Wiring fornisce la funzione **delay** per inserire un'attesa fra le esecuzioni di loop ed evitare di sovraccaricare il microcontrollore.

Il compilatore integrato nell'IDE compila il codice usato per lo sketch in C e lo inserisce all'interno della funzione main, usata come effettivo entry point al termine del bootloader, prima di ricompilarlo in codice oggetto per il dispositivo appropriato.

In sintesi, la struttura che uno sketch Arduino assume è la seguente:

```
void main() {

    ioinit();

    setup();

    while (1)
        loop();

}
```

La funzione **ionit**, chiamata prima di setup, prevede la configurazione di tutti gli ingressi analogici e digitali della scheda come input (di default) e altre operazioni di I/O.

Fra le altre cose l'IDE offre la possibilità di caricare un bootloader alternativo sulla scheda, consentendo eventualmente di impostare una diversa struttura logica per il codice eseguito.

# Capitolo 3

# Android

Android nasce come sistema operativo progettato per dispositivi mobili, quali smartphone, tablets ed e-readers, basato sul kernel Linux (versione 2.6, inizialmente) e di natura completamente open source, tutelato da licenza Apache.

## 3.1 Storia

Il progetto Android nasce con la società Android Inc., fondata nel 2003 a Palo Alto, in California, da Andy Rubin, Rich Miner, Nick Sears e Chris White.
La società nasce con lo scopo di progettare piattaforme per dispositivi mobili in grado di interagire del tutto autonomamente con l'utente, in base alle sue preferenze e posizione. Principalmente per problemi finanziari, i proprietari vendono la società a Google nel 2005 e si uniscono ad un suo team di sviluppo orientato allo stesso obiettivo.

Il 5 Novembre del 2007 nasce la **Open Handset Alliance**: un consorzio di compagnie che lavorano in ambito tecnologico quali software companies (Google, eBay), operatori di telefonia mobile (Sprint Nextel, Telecom Italia), produttori di smartphone (HTC, LG) e semiconduttori (Intel, NVIDIA).

La Open Handset Alliance si propone lo scopo di sviluppare uno standard aperto per dispositivi mobili, materializzato il giorno stesso nel loro primo prodotto: Android, una piattaforma per dispositivi mobili costruita sulla versione 2.6 del kernel Linux.
Meno di una settimana dopo viene rilasciato il primo Software Development Kit (SDK) per Android: una raccolta di strumenti per lo sviluppo di applicazioni Android, librerie, documentazione, tutorial e progetti di esempio, nonché un emulatore della piattaforma installabile su qualsiasi architettura x86.
Il primo smartphone in commercio con sistema operativo Android (versione 1) è l'HTC Dream, rilasciato sul mercato a partire dal 22 Ottobre 2008.
La prima versione del sistema presenta funzionalità quali: Android Market, per scaricare e aggiornare applicazioni pubblicate online, supporto per connettività wifi e bluetooth, un Media Player per riprodurre files multimediali, numerose utility Google (client di posta Gmail, Google Maps, ...) e tanto altro.

Segue uno storico delle release più importanti con relative features introdotte, a partire dalla 1.0 su HTC Dream fino all'ultima versione:

#### 1.0 - 23/09/2008

Primo rilascio ufficiale del sistema operativo.

#### 1.1 Petit Four - 09/02/2009

Update delle API e fix di numerosi bugs.

#### 1.5 Cupcake - 30/04/2009

Passaggio al kernel Linux 2.6.27. Aggiunta di numerosi servizi Google e widgets.

1.6 Donut - 15/09/2009

Passaggio al kernel Linux 2.6.29. Funzionalità di sintesi vocale e gesture. Aggiunta ricerca vocale e testuale dei contenuti presenti in locale e sul Web.

2.0 Eclair - 26/10/2009

Aggiunte funzionalità alla fotocamera, tra le quali zoom digitale. Aggiunta possibilità di sincronizzare più account Google e supporto agli account Exchange. Aggiunto il supporto al multi-touch e ai live wallpaper. UI e prestazioni migliorate.

2.2 Froyo - 20/05/2010

Passaggio al kernel Linux 2.6.32. Notevole miglioramento prestazionale grazie ad aggiunta di Compilazione Just In Time (JIT). Funzionalità di tethering USB e wifi. Integrazione del motore JavaScript V8 di Google Chrome nel browser di sistema. Supporto alla tecnologia Adobe Flash. Autoupdate delle applicazioni. Aggiunta di un Task Manager nativo per gestione processi.

2.3 Gingerbread - 06/12/2010

Passaggio al kernel Linux 2.6.35. Rivisitazione dell'interfaccia grafica. Ulteriore ottimizzazione del compilatore JIT. Aggiunto il supporto agli schermi XL (risoluzione WXGA e superiori). Integrate funzionalità SIP/VoIP e tecnologia Near Field Communication (NFC). Aggiunto supporto multi-touch a tastiera e migliorate funzionalità di copia/incolla e precisione adattiva. Facilitata la gestione di download di risorse HTTP tramite aggiunta di Download Manager. Supporto nativo per sensori come giroscopio e barometro. Migliorata la gestione energetica. Favorito il supporto di risoluzioni per tablet, aggiungendo l'attributo xlarge all'SDK.

### 3.0 Honeycomb - 22/02/2011

Passaggio al kernel Linux 2.6.36. Ottimizzata visualizzazione su tablet. Introdotta l'interfaccia grafica: Holo; comprensiva di una nuova barra di sistema (pulsanti software Home, Indietro, Task manager) e Action Bar (serie di opzioni dipendenti dall'applicazione aperta). Nuova visualizzazione multi-tab del browser di sistema. Aggiunto supporto per processori multi-core e rendering su GPU. Funzionalità di cifratura dei dati personali.

### 4.0.1 Ice Cream Sandwitch - 19/10/2011

Passaggio al kernel Linux 3.0.1. Opzione di wifi-direct per collegare direttamente lo smartphone a dispositivi dotati della stessa interfaccia. Nuovo approccio di sicurezza del dispositivo: Face Unlock, per sbloccarlo tramite riconoscimento facciale. Nuova interfaccia grafica comprensiva di pulsanti virtuali (per dispositivi privi dei corrispettivi fisici), cartelle su schermata home, un nuovo font di sistema (Roboto) e launcher personalizzabili. Aggiornate tutte le applicazioni di sistema per sfruttare le nuove API. Possibilità di scattare screenshots integrata nell'OS. Dettatura in tempo reale. Possibilità di eseguire alcune applicazioni senza sbloccare lo schermo. Fotocamera migliorata con ritardo di scatto nullo (zero shutter lag), modalità panorama e zoom durante la ripresa di video. Aggiunta integrazione diretta dell'applicazione contatti con vari social network. Applicazione di controllo del traffico di rete.

### 4.1.1 Jelly Bean - 27/06/2012

Passaggio al kernel Linux 3.1.10. Migliorie nell'utilizzo della tastiera: dizionari più completi, autocompletamento e voice typing più efficienti. NFC ottimizzato per condivisione di contenuto multimediale fra dispositivi. Personalizzazione della home e widgets più fluida. Ottimizzato l'utilizzo della CPU. Nuovo servizio Google Now.

Nel Settembre del 2012 Google ha mantenuto dati statistici relativi al numero di dispositivi che hanno effettuato l'accesso su Google Play (store online di Google) in una finestra temporale di 14 giorni.
I risultati delle rilevazioni sono sintetizzate nel seguente grafico:

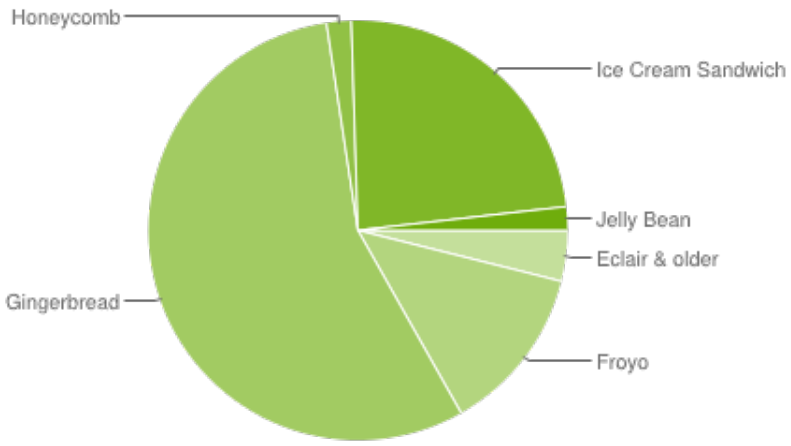



Figura 3.1: Grafico a torta per la distribuzione delle versioni del sistema operativo sui dispositivi in commercio

Ne risulta che Gingerbread, più precisamente nella versione 2.3.3 (API livello 10), rimane la versione della piattaforma più utilizzata fra quelle in circolazione.

## 3.2 Architettura

Il sistema Android si articola su una struttura gerarchica a diversi livelli di astrazione. Segue una breve descrizione di ogni livello: a partire dal kernel Linux alla base fino al soprastante livello applicativo.

### Kernel Linux

L'intera piattaforma Android è costruita al di sopra del kernel Linux 2.6, al quale sono state apportate alcune modifiche architetturali da Google per renderlo più adatto all'esecuzione su dispositivi mobili.

Il kernel Linux viene utilizzato fondamentalmente per ragioni di portabilità (essendo relativamente facile la compilazione su diverse architetture) e sicurezza: è stato testato su diversi ambienti operativi e presenta robusti meccanismi di controllo.

Questo livello è responsabile della comunicazione diretta con l'hardware del dispositivo, integrando al suo interno i drivers per il display, tastiera, wifi e altro.
Inoltre tutte le funzionalità base del sistema operativo, compresa la gestione dei processi e della memoria centrale, sicurezza e networking, sono svolte da questo nucleo.

## Librerie

Lo strato successivo è composto dalle librerie native di Android. Queste abilitano il device a gestire diversi tipi di dati, sono scritte in linguaggio C o C++ e sono specializzate per particolari hardware.
Alcune delle più importanti includono:

- Surface manager: componente responsabile della visualizzazione grafica, mantiene un off-screen buffer per componenti non immediatamente presenti su schermo.
- Media frameworks: una collezione di codecs per la registrazione e riproduzione di diversi formati multimediali.
- SQLite: database relazionale integrato nel sistema, utilizzato per memorizzazione di dati persistenti (privati per le singole applicazioni o globali).
- WebKit: browser engine utilizzato per visualizzare contenuto HTML.
- OpenGL: gestione di grafica 3D.

## Android Runtime

Questo livello comprende fondamentalmente la **Dalvik Virtual Machine** e le librerie Java fondamentali.

La Dalvik Virtual Machine, sviluppata da Dan Bornstein, consiste in una particolare Java Virtual Machine ottimizzata per poca potenza di calcolo e scarsa disponibilità di memoria.
In compilazione, i files .class (nel portabile Byte Code generato dalla JVM) vengono ricompilati da un compilatore Dex come Dalvik Byte Code, al fine di minimizzare le risorse utilizzate e aumentare l'efficienza. Viene inoltre generata una VM per ogni processo in esecuzione, al fine di garantire un maggiore isolamento e quindi sicurezza, specialmente per l'accesso alla memoria.

Le librerie Java fornite sono differenti da quelle appartenenti alle SE (Second Edition) e ME (Micro Edition, anch'esse utilizzate per dispositivi mobili), escludendo noti package quali AWT o Swing, ma conservando la maggior parte di quelli noti agli sviluppatori.

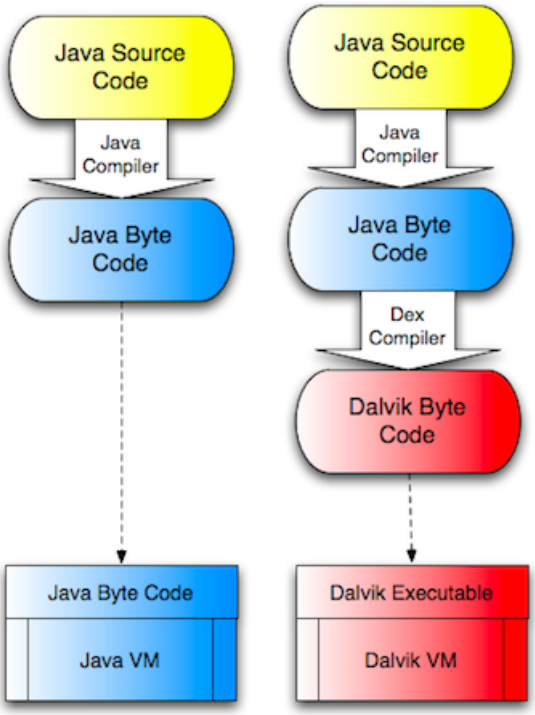



Figura 3.2: Differenze nella compilazione per Java Virtual Machine e Dalvik Virtual Machine

## Application Framework

Lo strato dell'Application Framework è notevolmente importante a livello di sviluppo: comprende infatti tutte le funzionalità offerte dal sistema e direttamente accessibili dalle applicazioni sviluppate (con opportuni wrappers).
Tra queste possiamo considerare:

- Activity Manager: gestisce il ciclo di vita delle applicazioni.
- Telephony Manager: gestisce le chiamate vocali.
- Content Providers: favoriscono la condivisione di dati fra applicazioni, fornendo un meccanismo per accedere ad essi.
- Location Manager: sfrutta il GPS o le network cells per localizzare geograficamente il dispositivo.
- Resource Manager: gestisce le risorse di natura eterogenea utilizzate nelle applicazioni.

## Livello delle applicazioni

Il livello più alto nell'architettura è quello che comprende le applicazioni vere e proprie, che utilizzano gli strati sottostanti (gestore contatti, calcolatrice, ...).
A questo livello non vi sono vere limitazioni nell'utilizzo delle risorse sottostanti: è sufficiente che l'applicazione dichiari a quali vuole accedere in lettura o scrittura. L'utilizzatore finale viene notificato circa i permessi che l'applicazione ha richiesto e spetta a lui decidere se utilizzarla o meno.
Questo sistema, assieme alla flessibilità di Google nei controlli delle applicazioni pubblicate su Market, ha sollevato numerose critiche e perplessità circa la sua sicurezza: è effettivamente stato riscontrato un numero notevole di applicazioni malware pubblicate, tra cui tentativi di phishing per mezzo di finte applicazioni bancarie.

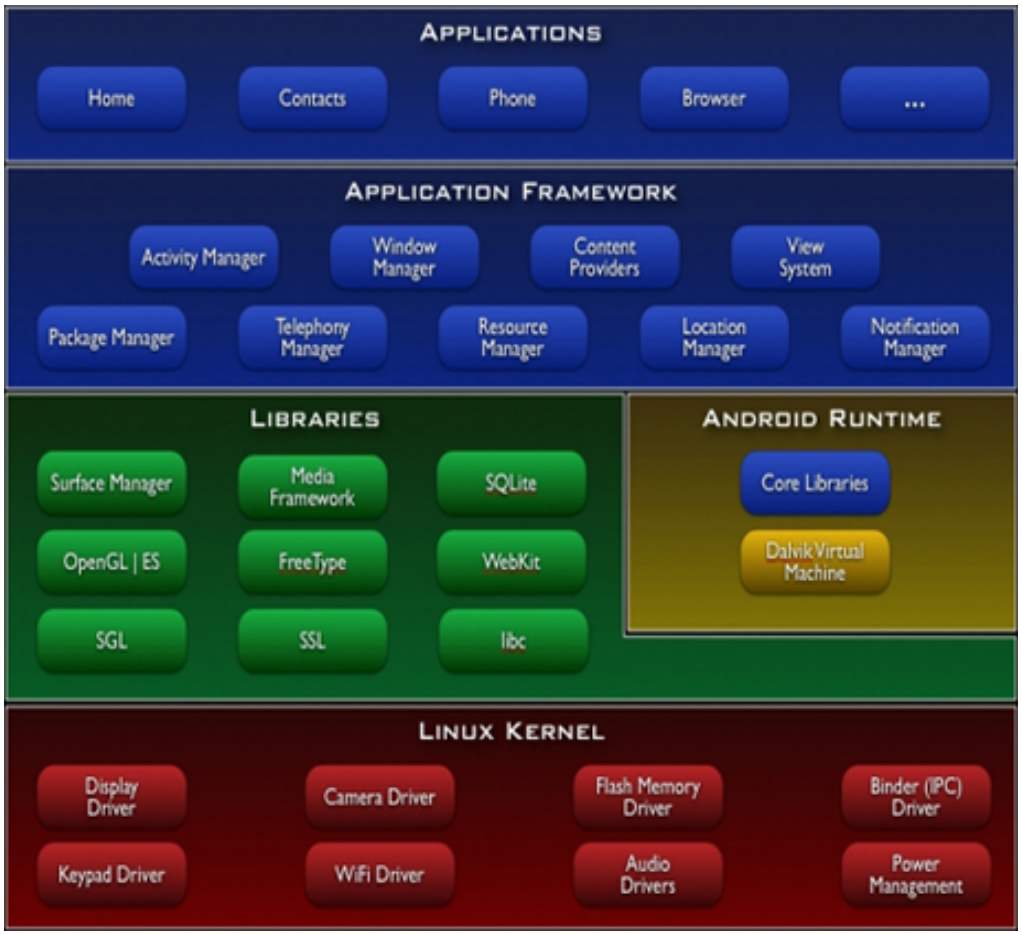



Figura 3.3: Architettura a strati del sistema Android

# Capitolo 4

# Integrazione di Android e Arduino

Nel corso di questi ultimi anni sono stati sviluppati numerosi progetti di domotica, principalmente a opera di amatori, che implementano un'interazione tra i microcontrollori Arduino e dispositivi mobili con sistema Android.
Tra questi sono compresi sistemi di controllo remoto di elettrodomestici, rilevazione e pubblicazione sul web di dati ambientali, movimento di modellini mobili a motore e supporto alla sicurezza di edifici.

Probabilmente il progetto non ufficiale di maggior successo è **Amarino**: un toolkit, comprensivo di strumenti hardware e software, che consente la comunicazione tra Android e Arduino mediante USB o bluetooth.
Il pacchetto software offerto comprende un'applicazione installabile su piattaforma Android e una serie di librerie per Arduino. Sfruttando una combinazione delle due è possibile comunicare direttamente via bluetooth con diversi dispositivi connessi alla scheda Arduino, con funzionalità avanzate di rilevazioni eventi (come il valore di un sensore direttamente notificato su smartphone) e attivazione remota.
L'associazione open source **Build Circuit** ha inoltre rilasciato un kit di accessori per Arduino, adatti all'uso con gli sketch di esempio nelle librerie offerte da Amarino, quali resistenze, LED e alcuni sensori.

Un secondo progetto di integrazione tra Android e Arduino è stato promosso direttamente da Google, in occasione dell'evento Google I/O 2011: **Android Open Hardware**. Questo progetto include un Android ADK (Accessory Development Kit) e un protocollo di comunicazione via USB o bluetooth con periferiche audio e altri accessori: Android Open Accessory Protocol.

L'Android ADK consiste in:

- Una scheda Arduino basata sul microcontrollore ATmega2560 e denominata Arduino ADK, dotata di un ingresso USB per comunicare con dispositivi basati su Android e un convertitore boost per alimentarli direttamente. La versione rilasciata nel 2012 comprende anche un lettore di micro SD e numerosi sensori integrati (temperatura , luce, prossimità, umidità, ...).
- Un ambiente di sviluppo integrato (IDE) per sviluppare sketch da eseguire sull'Arduino ADK, focalizzato su comunicazione con dispositivi Android.
- Un framework Android per facilitare la comunicazione con l'Arduino ADK e sue periferiche, via USB o bluetooth.

Questo progetto è di natura open-source: il design della scheda, il codice sorgente del suo firmware e quello dell'IDE sono inclusi nel pacchetto software, disponibile su web.

# Parte II

# Implementazione

# Capitolo 5

# Strumenti

Segue una descrizione degli strumenti Hardware e Software impiegati nell'implementazione del progetto.

Questi comprendono le schede Arduino (due diversi modelli) utilizzate per processare i dati rilevati dai sensori e collezionarli in un formato di facile memorizzazione ed elaborazione.
Le informazioni sono scambiate tra i componenti della rete mediante uso di moduli Xbee (una particolare piattaforma che sfrutta i protocolli di rete 802.15.4 / ZigBee ), integrati con ogni scheda Arduino tramite Xbee Shield.

Per lo sviluppo su piattaforma Android è stato fatto uso principalmente dell' Android Virtual Device offerto con lo SDK, nonché di uno smartphone modello HTC Wildfire S, al fine di ovviare a certe limitazioni presenti nell'emulatore.
Le versioni del sistema operativo utilizzate sono la 2.3.3 (Gingerbread) e 4.1 (Jelly Bean).

Per quanto riguarda la codifica si è fatto uso del tool X-CTU per modificare il firmware dei moduli XBee, l'IDE ufficiale per programmazione su schede Arduino e l'ambiente integrato per Android dell'editor Eclipse per la programmazione su smartphone.

## 5.1 Hardware

Ogni componente Hardware utilizzato per la realizzazione del progetto viene descritto brevemente in questa sezione.

### Moduli XBee

I modem radio XBee sono prodotti dalla Digi International in versioni Standard e Pro, per usi avanzati, e sfruttano l'insieme di protocolli di comunicazione ZigBee.

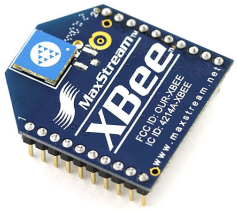



Figura 5.1: Un modulo XBee

Nello specifico quelli utilizzati per realizzare la comunicazione via radio fra le schede corrispondono al modello XBee 2 (o XBee Series 2) Standard.
Questi modelli hanno un raggio d'azione compreso tra i 40 metri (in ambienti interni) e 120 (esterni), una potenza di trasmissione in output di 2 milliWatt e un data rate di trasferimento pari a circa 250 Kbps.

Il Firmware integrato include i protocolli ZigBee, soprastanti i livelli PHY e MAC (a differenza dei firmware della prima serie) ed eccelle in scenari che prevedono un basso consumo energetico, quindi una rete di sensori.
I modelli della seconda serie comprendono le due modalità di comunicazione menzionate nel capitolo 1.3: API (assente nella prima serie) e trasparente.

La configurazione del Firmware dei moduli è stata effettuata mediante un **XBee Explorer Dongle** (adattatore USB-Serial), per comunicazione seriale tra un generico modulo ed un computer.

### Schede Arduino

Sono state utilizzate due diverse versioni della scheda Arduino, in base al ruolo ricoperto da un nodo nella rete:

- Diversi **Arduino Uno**, designati come End Device nella rete di sensori.

  Le specifiche tecniche di queste schede comprendono 14 pins digitali di input/output e 6 analogici per input, una presa USB per alimentazione e programmazione, una memoria Flash da 32 KB (0.5 usati dal bootloader), SRAM da 2 KB ed EEPROM da 1 KB.

- Un singolo **Arduino Ethernet**, utilizzato come Coordinator all'interno della rete ZigBee e mediatore tra questa e i dispositivi esterni.

  Questo controllore è effettivamente basato sull'Arduino Uno, integrando però un ethernet controller W5100 con stack di protocolli TCP/IP, un socket per l'aggiunta di un modulo PoE (Power over Ethernet) e un lettore di schede microSD.

La comunicazione fra entrambi i tipi di dispositivi e i moduli XBee avviene tramite un adattatore tra i circuiti delle due interfacce: un **XBee Shield**.

Collegato ai connettori esposti sulla scheda Arduino, lo Shield mette in diretta comunicazione il pin DOUT (UART Data Out) del modulo Xbee con il pin RX del microcontrollore e il pin DIN (UART Data In) con TX.
In questo modo i dati scritti sulla porta seriale della scheda vengono direttamente letti dal modulo XBee e spediti sulla rete e quelli ricevuti vengono immagazzinati nel buffer di ricezione del microcontrollore.
Questo comportamento è regolato da due jumpers rimovibili, che mettono in comunicazione i sopracitati pins di uscita e ingresso (rimuovendoli la stessa porta viene utilizzata per comunicazione via USB e quindi programmazione della scheda).

Questa interfaccia predispone il modulo XBee all'uso della modalità trasparente di comunicazione, semplificando la comunicazione e riducendo l'overhead derivante dall'invio di frames di dati con un formato più complesso (si veda 6.2).


Figura 5.2: Xbee Shield

È opportuno segnalare l'esistenza di una versione ottimizzata di questo accessorio: il Wireless Proto Shield.
Quest'ultimo offre maggiori vantaggi: una circuiteria semplificata, il supporto di diversi tipi di moduli (wifi e bluetooth in aggiunta) e uno switch per passare dalla comunicazione USB a quella con il modem connesso.
Tuttavia per ragioni di disponibilità si sono utilizzati gli Xbee Shield.

## 5.2 Software

Per la codifica, il debugging e testing dei vari moduli software che compongono il progetto sono state utilizzate diverse piattaforme software.

### X-CTU

X-CTU è un tool gratuito e distribuito dalla Digi International per interagire con i suoi prodotti.

Questo strumento consente di leggere il firmware dei moduli XBee (connessi via USB attraverso un XBee Explorer Dongle) e sostituirlo con una delle versioni integrate (alcune specifiche per un particolare utilizzo dei dispositivi), eventualmente effettuando il download delle nuove direttamente dal sito web della Digi.
Con il firmware è possibile anche cambiare la modalità di comunicazione del modulo (API o trasparente). I parametri utilizzati durante la connessione ad una rete possono essere letti e scritti usando un'interfaccia a terminale.

Inoltre è prevista una utile funzionalità di test della comunicazione di un nodo con la sua rete: potenza del segnale e percentuale di pacchetti persi su quelli ricevuti o spediti.

### Arduino IDE

Per la programmazione degli sketch in esecuzione sulle schede Arduino e per caricarvi il codice è stato utilizzato l' Arduino Development Environment ufficiale.

Il software consiste principalmente in un editor per la codifica, con semplici funzioni di indentazione automatica, syntax highlighting e verifica sintattica del codice.
È possibile selezionare diversi profili di dispositivi su cui trasmettere il codice sviluppato, onde permettere ottimizzazioni basate sugli specifici hardware; diversi sketch possono essere visualizzati su più tabs e caricati su schede collegate agli ingressi USB del computer in uso.

L'IDE integra al suo interno numerose librerie per facilitare la comunicazione via rete (la libreria Ethernet) e la lettura di segnali analogici o digitali esterni.

In aggiunta a queste è stata utilizzata una libreria esterna per manipolare dati in formato JSON: **aJson**, scritta da Marcus Nowotny.
Per ulteriori riferimenti si rimanda alla pagina del progetto:
http://interactive-matter.eu/blog/2010/08/14/ajson-handle-json-with-arduino/

## Eclipse

Per lo sviluppo dell'applicazione smartphone su piattaforma Android è stato utilizzato l'IDE Eclipse, in congiunzione con l'Android SDK e l'Android Virtual Device Manager per creare e configurare dispositivi virtuali.

Sebbene esistano numerose alternative, Eclipse rappresenta probabilmente il miglior ambiente di sviluppo per scrivere l'applicazione, grazie ad un editor eccellente e una perfetta integrazione con le funzionalità offerte dall'SDK.

La maggior parte dei test di esecuzione, durante la fase di codifica, sono stati eseguiti su due diverse istanze dell'emulatore: una con la versione 2.3.3 del sistema operativo, l'altra con la versione 4.1 (a oggi la più recente); questo allo scopo di verificare l'effettiva retrocompatibilità con una delle versioni più diffuse (Gingerbread).
Fra le tante limitazioni di cui l'emulatore soffre, la mancanza di un supporto per connessione wifi, fondamentale nella costruzione di questa applicazione, ha reso necessario l'utilizzo di una piattaforma fisica su cui effettuare i test: uno smartphone modello HTC Wildifire S, con la versione 2.3.3 del sistema operativo installata.

# Capitolo 6

# Rete

La rete di sensori costituisce il primo dei due sottosistemi che compongono questo progetto.

Il meccanismo retrostante l'interfaccia che questa rete offre deve essere completamente nascosto all'esterno, allo scopo di favorirne la comunicazione con applicazioni di natura eterogenea: non necessariamente mobili e in esecuzione su dispositivi smartphone, ma anche su personal computer, server e altre infrastrutture.

Questo sistema si può articolare su due livelli di astrazione: quello dei modem XBee, configurati per formare una rete e trasmettersi informazioni secondo un determinato schema, e quello dei microcontrollori che generano ed elaborano queste informazioni, ignari dei protocolli di comunicazione via radio che li mettono in reciproca comunicazione.

## 6.1 Architettura

La rete, costruita utilizzando 5 moduli XBee, assume una semplice topologia a stella.

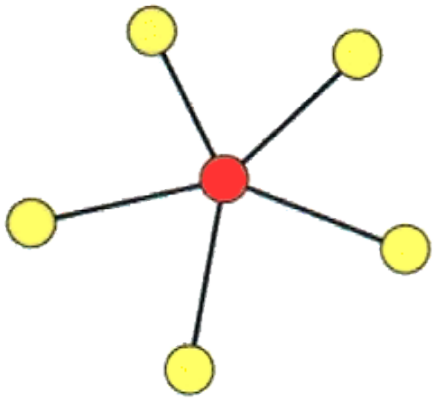


Figura 6.1: Modello a Stella

Questa topologia di rete prevede un unico gateway centrale (il Coordinator, in questo caso) al quale fanno riferimento tutti i nodi partecipanti, ossia i 4 End Device. In una rete di queste dimensioni non sono necessari nodi che assumano il ruolo di Router.

La comunicazione fra gli End Device e il Coordinator è unidirezionale, semplificandone notevolmente la gestione e minimizzando il traffico dovuto al routing dei pacchetti o il mutuo scambio di messaggi fra due o più nodi.

## 6.2 Comunicazione

Il firmware dei moduli utilizzati corrisponde alla versione **XB24-ZB**, con un set di parametri e funzioni diverse per il Coordinator e gli End Device.

La modalità di comunicazione prevista dal firmware scelto è quella trasparente (denominata anche modalità AT); come precedentemente accennato, usando questa modalità il dispositivo che interagisce con il modulo (il microcontrollore Arduino, in questo specifico caso) può astrarsi completamente dalle sue specifiche e trasmettere il semplice corpo dei dati.
Infatti, il singolo modulo XBee è semplicemente configurato con l'indirizzo del destinatario di tutti i suoi messaggi, rendendo la comunicazione a senso unico.
Tutti i dati ricevuti nel buffer di ingresso del modulo, quindi attraverso il pin DIN, vengono automaticamente trattati come il payload di un frame dati e spediti di conseguenza con un formato fisso.

Usando invece la modalità API del Firmware, il controllore sarebbe costretto ad inserire l'header dei messaggi nel flusso di dati serializzati destinato al modem, specificando informazioni quali il destinatario e la lunghezza del messaggio.

L'evidente considerazione sarebbe che, nonostante il costo derivante dal dover creare l'intero frame comprensivo di header, la modalità API permetta una maggiore flessibilità nella comunicazione, consentendo possibilità quali la selezione dinamica del destinatario di un messaggio.

La natura stessa della rete di sensori, nella quale è previsto che dispositivi che raccolgono dati esterni (quindi connessi a modem End Device) siano alimentati a batteria e comunichino le loro informazioni ad un unico nodo centrale, rende tuttavia più indicato l'uso della modalità AT.

Poiché il limite di nodi figli supportato da un Coordinator che usa questo firmware è pari a 10, qualora fosse necessaria un'estensione della rete ad un numero maggiore di nodi si potrebbe convertirla in una a topologia Cluster Tree:

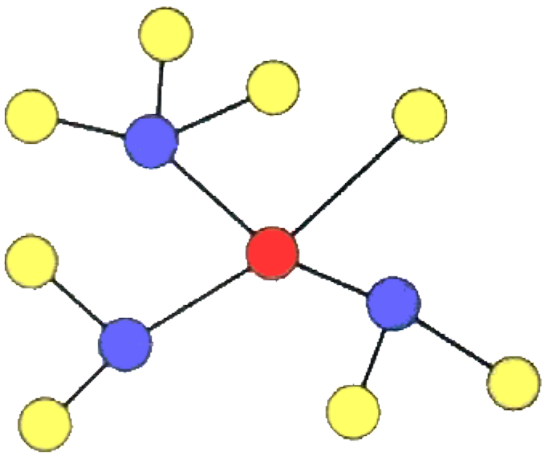


Figura 6.2: Modello a Cluster Tree: i nodi blu rappresentano i Router

Questo schema prevede l'introduzione della terza tipologia di nodo ZigBee: il Router.

Ogni End Device che non sia direttamente connesso al Coordinator invia i propri dati al suo Router, il quale si limita a inoltrarli al Coordinator.

Questo permette di aumentare il numero di nodi nella rete (aumentando il numero di Hops compiuti da un frame di dati nel raggiungere il Coordinator) fino al limite fissato dallo spazio di indirizzamento a 16 bit previsto per il Network Address ZigBee.
Il Router ha inoltre una capacità (in termini del numero di figli) leggermente superiore a quella del Coordinator: 12.

Il sistema presenta anche un certo grado di Fault Tolerance: se in una rete composta da più Router uno di questi si guasta, il livello Network del protocollo ZigBee prevede che gli End Device precedentemente connessi ad esso effettuino un Orphan Scan (si veda 1.2.1) per trovare un nuovo padre.

Il microcontrollore connesso al modem, da parte sua, è costretto ad attivarne la Command Mode (scrivendo i caratteri ASCII +++ sul pin DIN del modulo) e controllare l'identificativo del nodo padre, se questo è presente, prima di inviare ogni dato; in questa maniera, se necessario, potrà reimpostare la nuova destinazione del modulo.
Ad ogni modo, ad un costo minimo (pochi scambi di dati fra i due dispositivi) è possibile evitare che il guasto di un Router comprometta un intero Cluster della rete.

## 6.3 Coordinator

La sezione che segue descrive in maggiore dettaglio la configurazione di un nodo per farlo operare come Coordinator della rete.
Verranno omessi i particolari concernenti la sequenza di azioni che il modem esegue per creare la rete e accettare nuovi nodi (si veda 1.2.1), concentrandosi piuttosto sulla descrizione dei principali comandi forniti e di come questi delineino un determinato comportamento del modem.

La configurazione del modulo avviene tramite la funzionalità Terminal del tool X-CTU. Ogni comando è preceduto dal prefisso AT e il modulo replica con un OK se la scrittura dei parametri è avvenuta con successo.
Il salvataggio dei parametri su memoria persistente viene indotto mediante il comando ATCN.

```
+++OK
```

Usando la modalità di comunicazione trasparente è necessario inizializzare la Command Mode del modem (fase in cui i caratteri ricevuti sono interpretati come comandi di scrittura e lettura dei parametri del firmware), scrivendo i simboli ASCII +++ e attendendo l'OK del modulo.
Si ricorda che le stesse sequenze comando-risposta possono essere eseguite direttamente dal microcontrollore che utilizza il modulo, al fine di riconfigurarlo dinamicamente e in maniera automatica.

La rete formata dal Coordinator sarà identificata univocamente da una coppia di parametri: canale utilizzato e PAN ID.
Poiché il Coordinator, in fase di inizializzazione, effettua un Energy Scan per determinare il miglior canale su cui trasmettere, è possibile impostare tramite una bitmap una lista di canali tra cui gli è concesso scegliere.

In combinazione con un PAN ID statico, vale a dire che se su un determinato canale esiste una rete con quel PAN ID, il Coordinator è costretto a sceglierne un altro, è possibile forzarlo a creare una rete con un determinato identificativo. Questo può essere necessario qualora siano presenti più reti ZigBee nella stessa area.

```
ATSC FFFF
OK
ATID F
OK
```

Usando la bitmask FFFF, impostata con il comando SC, si indica al Coordinator di controllare tutti i canali possibili, cercando il migliore su cui trasmettere.
In un contesto con poche reti ZigBee coesistenti il PAN ID statico (F, in questo caso) è sufficiente a evitare che gli End Device si connettano alla rete sbagliata.

```
ATNJ FF
OK
```

Il parametro Node Join Time (NJ) indica al Coordinator il tempo, in secondi, per il quale deve consentire il join di altri nodi alla rete.
Questo parametro è necessario solo per ragioni di sicurezza, ma per favorire un certo grado di scalabilità alla rete, in cui è consentito aggiungere nodi senza doverla riconfigurare, viene settato con il valore FF, a indicare l'assenza di un limite di tempo.

```
ATDH 0
OK
ATDL FFFF
OK
```

I parametri DH e DL indicano rispettivamente i 32 bit più e meno significativi dell'indirizzo a 64 bit del nodo destinatario per i dati inviati.
Usando FFFF si indica di trasmettere in broadcast (un indirizzo unicast di un singolo nodo non avrebbe senso per il Coordinator, in una topologia di rete a stella).

```
ATBD 3
OK
ATNB 0
OK
```

Questi comandi riguardano la comunicazione seriale con il microcontrollore utilizzato.
Il comando BD setta la Baud Rate con la quale il dispositivo deve trasmettere (3 indica un valore di 9600) e NB a 0 indica di non usare un meccanismo di bit di parità sui dati trasmessi, poiché lo scenario previsto non costringe un'assoluta intolleranza agli errori di trasmissione.

## 6.4 End Device

La sezione che segue descrive in maggiore dettaglio la configurazione dei restanti nodi per farli operare come End Device all'interno della rete.

```
ATSC FFFF
OK
ATID F
OK
```

In analogia con il Coordinator, ogni End Device deve essere configurato per riconoscere i beacon relativi a reti con un PAN ID pari a F, sempre ammesso che lo scenario non comprenda numerose reti presenti nello stesso raggio d'azione.

```
ATDH 0
OK
ATDL 0
OK
```

Un destination address pari a 0 indica al device di trasmettere sempre al Coordinator della rete.
Usando questi parametri è possibile garantire che, in una rete di tipo Cluster Tree, se un Router dovesse guastarsi e tutti gli End Device connessi ad esso trovassero un nuovo Router disponibile, questi inoltrerebbe automaticamente tutti i pacchetti ricevuti al Coordinator.

```
ATSM 5
OK
ATST 3E8
OK
ATSP AF0
OK
```

I rimanenti parametri sono relativi alla modalità di sleep dell'End Device, particolarmente significativa in una rete di sensori a basso consumo energetico.
Il parametro SM indica la modalità selezionata: un valore di 5 corrisponde ad uno sleep ciclico, combinato con la possibilità di essere risvegliato con l'asserzione del pin Sleep-RQ (pin 9 sul modulo) dal microcontrollore.
I restanti parametri indicano rispettivamente il tempo di inattività necessario a portare il modulo in sleep (in millisecondi) e la durata del periodo di sleep (in decine di millisecondi).
Naturalmente il valore scelto per ST e SP andrebbe adattato alla frequenza con cui i microcontrollori elaborano e trasmettono le informazioni rilevate.

La scelta non è ricaduta sulla modalità Pin Hybernate (valore 1 di SM), che costringe il sistema host ad addormentare e risvegliare il modem asserendo il pin Sleep-RQ, per rendere il microcontrollore indipendente da questo meccanismo e favorire una maggiore scalabilità della rete.

# Capitolo 7

# Microcontrollori

Il livello soprastante nella rete di sensori è costituito dai dispositivi Arduino che processano le informazioni da trasmettere.
Sfruttando l'utilizzo della modalità trasparente di comunicazione dei moduli XBee, le schede Arduino possono astrarsi completamente dalle interfacce radio che li mettono in reciproca comunicazione, limitandosi a scrivere e ricevere i dati sull'interfaccia seriale:

```
Serial.begin(9600); // Baud Rate usata dal modulo
Serial.print(msg);
```

Scrive i dati nel buffer di input del modulo XBee integrato tramite XBee Shield, usando l'interfaccia UART, in modo che questo li invii alla destinazione preimpostata.

```
byte b = Serial.read();
```

Legge un byte dal buffer di output del modulo XBee.

I ruoli assunti dalle singole schede riflettono quelli dei modem XBee annessi:

- Un unico nodo centrale, denominato **Sink** (pozzo).

  Questo dispositivo è formato dall'Arduino Ethernet con XBee Shield annesso e ha lo scopo di raccogliere le informazioni ricevute dagli altri nodi, aggregarle secondo un criterio preciso e, periodicamente, spedirle in broadcast sull'interfaccia ethernet, in modo che eventuali applicazioni in ascolto su essa possano riceverle e processarle. Idealmente questo nodo è l'unico a necessitare di alimentazione elettrica.

- Una serie di nodi End Device, alimentati a batteria e connessi a sensori o attuatori; questi dispositivi consistono negli Arduino Uno con Xbee Shield e si limitano a raccogliere dati con una determinata frequenza e spedirli al nodo Sink.

## 7.1 End Device

Le operazioni svolte da un generico End Device appartenente alla rete consistono fondamentalmente nel leggere i dati dai sensori, costruire un messaggio contenente le informazioni che descrivono questi dati, spedirli e dormire per un certo lasso di tempo.

La prima di queste operazioni dipende strettamente dal tipo di sensore utilizzato: potrebbe essere il calcolo dell'equazione di Steinhart-Hart sulla tensione prodotta da un Thermistor, per rilevazione di temperatura, o la semplice lettura dei valori digitali prodotti da uno Switch.

Per la costruzione del messaggio si utilizza un protocollo particolare (descritto brevemente nella prossima sezione e approfondito nel capitolo 8.1) e il formato JSON, tramite la libreria esterna aJson.
Il formato JSON è stato scelto in quanto snello e di semplice parsing.

Il protocollo comprende un semplice sistema di Sequence Number dei messaggi (con uno spazio molto piccolo: 0-99, ossia due caratteri utilizzati nella rappresentazione testuale), utilizzato in sostituzione ad un più verboso timestamp, al fine di semplificare la gestione dei duplicati da parte dell'applicazione mobile comunicante con il Sink.

I passi necessari per la creazione di un messaggio sono esplicati dalla seguente funzione:

```
aJsonObject* makeMsg(char* value) {

  char currentSN[SN_MAX_DIGITS];

  itoa(sn,currentSN,10);

  char* keys[] = { ID_KEY, SN_KEY,
        VALUE_KEY, LOCATION_KEY, SCALE_KEY};

  char* values[] = { TEMPERATURE_ID, currentSN,
        value, LOCATION, USED_SCALE};

  aJsonObject* json = aJson.createObject();

  for (int i = 0; i < MSG_FIELDS; i++)
    aJson.addItemToObject(json, keys[i],
        aJson.createItem(values[i]));

  sn = (sn + 1) % MAX_SN;

  return json;
}
```

Il valore del sensore (in questo caso la temperatura) è passato come parametro alla funzione.

I due array di puntatori a carattere inizializzati conterranno rispettivamente le chiavi (nel formato previsto dal protocollo di comunicazione) e i valori, tra i quali vi sono:

- Un valore alfanumerico che rappresenta il tipo di dato, incapsulato nella costante TEMPERATURE_ID.
- La rappresentazione testuale di un intero, ottenuta mediante la funzione itoa, che esprime il Sequence Number abbinato al messaggio.
- Le due variabili location e value, puntatori a carattere di cui il primo costante e dichiarato come una variabile globale, rappresentano il luogo dove è collocato il dispositivo e il valore rilevato.
- Il quarto parametro descrive la scala utilizzata per esprimere la temperatura.

Viene successivamente creato l'oggetto aJsonObject e i suoi elementi inizializzati come coppie chiave-valore relative alle posizioni nei due array.
L'oggetto viene allocato su heap, permettendone la sopravvivenza alla distruzione del record d'attivazione della funzione.

Infine, prima di restituire un puntatore all'oggetto creato, il Sequence Number attuale relativo al messaggio spedito viene incrementato di uno, riportandolo a 0 se supera il valore della costante MAX_SN (99).

L'invio di un singolo messaggio, contenente i dati relativi alla temperatura rilevata, consiste nella chiamata a questa funzione:

```
void sendMsg() {

  char value[FIELD_MAXSIZE];

  itoa(getSensorValue(),value,FIELD_MAXSIZE);

  aJsonObject* data = makeMsg(value);

  char* msg = aJson.print(data);

  Serial.println(msg);

  aJson.deleteItem(data);
  free(msg);

}
```

Il dispositivo converte il valore rilevato dal sensore in una stringa, memorizzandola in un array di caratteri e passando l'array come parametro alla funzione makeMsg.
L'oggetto aJsonObject così ottenuto sarà successivamente convertito in stringa e trasferito via porta seriale al modem XBee, il quale lo invierà al Coordinator nella sua rete, ossia il Sink.
È necessario infine deallocare dallo heap l'oggetto aJsonObject e la sequenza di caratteri ottenuta da esso.

## Il protocollo

Trattandosi di dispositivi dotati di una quantità di memoria assai esigua (la quantità di SRAM di un Arduino Uno corrisponde a 2 KB) si è reso necessario l'utilizzo di un protocollo molto conciso per minimizzare la quantità di dati trasferiti.
I dati rilevati, nonché i metadati relativi ad essi (la collocazione del sensore, la scala utilizzata per la temperatura o il semplice modello del dispositivo) non possono essere rappresentati in un formato leggibile da essere umano, quindi fin troppo verboso, ma tramite una sequenza di caratteri alfanumerici.

Per esempio, la serie di dati:

{Tipo : Temperatura, Valore : 30, Collocazione : Stanza2, Scala : Celsius}

è in un formato estremamente verboso, nonché interpretato in una lingua specifica e quindi poco portabile.

Al contrario, la stringa:

{id : a-0, x0 : 30, x1 : Stanza2, x2 : Celsius}

può avere la stessa interpretazione, ma in un formato più stringato e, sebbene non human-readable, più portabile tra diverse piattaforme.

Per maggiori dettagli sulla definizione del protocollo e sua interpretazione dalla applicazione context-aware si veda il capitolo 8.1.

## 7.2 Sink

Lo sketch caricato sul nodo designato come Sink (quindi sull'Arduino Ethernet) assume una complessità decisamente maggiore: occorrerà da una parte ricevere i dati, controllarne la validità ed aggregarli in una struttura di memorizzazione, eventualmente sostituendoli a vecchie copie; dall'altra, periodicamente, raggrupparli per tipo ed inviarli in broadcast attraverso l'interfaccia ethernet.

Per ragioni di spazio, nel descrivere l'esecuzione si riportano solo gli spezzoni principali dello sketch.

In sostanza, la funzione loop esegue le seguenti istruzioni, tentando la ricezione di dati ogni 200 millisecondi ed inviando quelli collezionati sull'interfaccia ethernet ogni 15 secondi:

```
void loop() {
  if (ticks > 75) {
    broadcastData();
    ticks = 0;
  }
  else {
    ticks++;
    if (Serial.available())
      getData();
  }
  delay(200);
}
```

La funzione getData legge innanzitutto una sequenza di caratteri dal buffer della porta seriale, fino a ricezione del carattere di fine messaggio (il carattere di new line) o allo scadere di un timeout.

```
while (((c = Serial.read()) != -1
        || millis() - now < READ_TIMEOUT)
    && char(c) != MSG_SEPARATOR) {
    if (c != -1)
      buffer[pos++] = char(c);
}
```

Il vettore di caratteri usato come buffer viene successivamente convertito in una struttura JSON.
La funzione parse della libreria aJson incapsula al suo interno il controllo che la stringa non sia un JSON malformato (e quindi non se ne siano persi dei frammenti).

```
if ((data = aJson.parse(buffer)) != NULL) { ... }
```

Se la funzione non restituisce NULL (quindi il JSON presenta un formato corretto) viene fatto un confronto fra le altre strutture memorizzate in precedenza, in un vettore di puntatori a aJsonObject memorizzati su heap, controllando specificatamente i campi id (descrizione del tipo di dato) e x1 (localizzazione).

Se il confronto ha esito positivo, quindi esiste un aJsonObject con gli stessi valori per le due chiavi, si effettua una sostituzione, cancellando il vecchio dato dallo heap.
Nel controllare questi campi ci si basa sull'assunzione che non vi saranno due dispositivi diversi che invieranno lo stesso tipo di dati dalla stessa collocazione (per esempio due Thermistor in posizioni ravvicinate rappresenterebbero una ridondanza).

Qualora si rivelasse necessario, le stringhe descrittive potranno comunque essere diverse per disambiguare le diverse rilevazioni (ad esempio: SalaServer1 e SalaServer2).
Si è ritenuto che fosse sufficiente come identificazione univoca, dato che questa situazione si presenterà piuttosto raramente e l'aggiunta di un ulteriore identificativo per un singolo dispositivo comporterebbe una maggiore quantità di memoria utilizzata.

Se lo spazio riservato alla memorizzazione delle strutture è esaurito si sostituisce la prima memorizzata in ordine di tempo, deallocandola, mediante cursore su una coda circolare:

```
if (stored[head] != NULL)
        aJson.deleteItem(stored[head]);

stored[head] = data;
data = NULL;

head = (head + 1) % MAX_DATA;
```

La variabile data rappresenta la nuova struttura, mentre le precedenti sono memorizzate nel vettore stored.
Si noti che, data la struttura a coda circolare utilizzata per memorizzare i dati, il Sink emetterà ciclicamente tutte le rilevazioni pervenute senza alcuna perdita di informazione.

È stata considerata anche la possibilità che si usasse l'identificativo

$$< Tipo\,di\,dato, Collocazione >$$

su dispositivi differenti e che il Sink calcolasse una media pesata sui valori ottenuti, a ricezione di un nuovo dato.

A questo scopo, una potenziale soluzione sarebbe quella di mantenere uno storico degli ultimi valori rilevati per ogni tipo e calcolarvi una semplice media aritmetica.
Questa soluzione non è accettabile proprio a causa della già menzionata scarsa capacità di memorizzazione dei dispositivi.
Un'alternativa sarebbe quella di calcolare la media con peso $\alpha$, traducibile nell'importanza attribuita alle nuove rilevazioni, fra il valore corrente e l'ultimo risultato ottenuto.

Quindi, dato $x$ nuovo valore rilevato e $avg$ media calcolata, il nuovo valore medio è dato da:

$$avg_n = avg_{n-1}(1 - \alpha) + x\alpha$$

Il fattore critico è la scelta di $\alpha$, che andrebbe adattato a seconda del contesto: un peso troppo piccolo comporterebbe il rischio di trascurare una variazione rilevante (si pensi a più fotoresistori nella stessa stanza buia e uno di essi posto sulla spia di un allarme); al contrario, un peso troppo grande causerebbe oscillazioni indesiderate nei valori rilevati.

In entrambi i casi, pur riducendo la mole di dati che il Sink inoltra sulla rete ethernet, si avrebbe una perdita di informazioni non trascurabile.
Si è quindi ritenuto opportuno che il Sink si limitasse a discriminare le informazioni ottenute in base a tipo di dato e collocazione del nodo, senza compiere ulteriori aggregazioni.

La seconda operazione fondamentale che il Sink compie è quella di inviare i dati in broadcast sull'interfaccia ethernet.
Per questo sono state utilizzate le librerie Ethernet ed EthernetUDP.
In fase di setup viene acquisito un indirizzo IP tramite chiamata della funzione Ethernet.begin, passando come parametro il MAC Address dell'Arduino Ethernet utilizzato.

Il primo passo consiste nel raggruppare i dati dello stesso tipo, quindi con chiave id all'interno della struttura JSON, controllando che non si superi una certa soglia (ossia la dimensione del buffer di trasmissione in memoria); se tale soglia fosse oltrepassata, la stringa sarà divisa in due parti.

La stringa così ottenuta viene incapsulata nel payload di un pacchetto UDP e spedita in broadcast attraverso la rete, con un breve delay per permettere a dispositivi più lenti di elaborare i pacchetti, nel caso di un burst derivante da molti tipi diversi:

```
udp.beginPacket(address, port);
udp.write(buffer);
udp.endPacket();

delay(200);
```

Vale la pena menzionare che per ovviare alla scarsa disponibilità di memoria si è fatto uso della libreria pgmspace, al fine di salvare su memoria flash alcune stringhe costanti, come quella che descrive l'indirizzo di broadcast nella rete.
Qualora si rendesse necessaria la memorizzazione di dati per numerosi dispositivi occorrerebbe fare uso della memoria EEPROM o di una scheda micro SD, per cui l'Arduino Ethernet ha un lettore integrato.

La seguente foto mostra tre delle schede usate per formare la rete.

Il dispositivo etichettato come End Device 1 accende o spegne un LED verde.
Il secondo End Device, tramite uso di una breadboard, accende o spegne un LED rosso e legge il valore di un fotoresistore; questa scheda, a differenza delle altre due, fa uso di un Wireless Proto Shield per interfacciarsi con il modulo XBee.
Il Sink riceve i dati dagli altri dispositivi e li emette in broadcast sull'interfaccia di rete ethernet a cui è connesso.

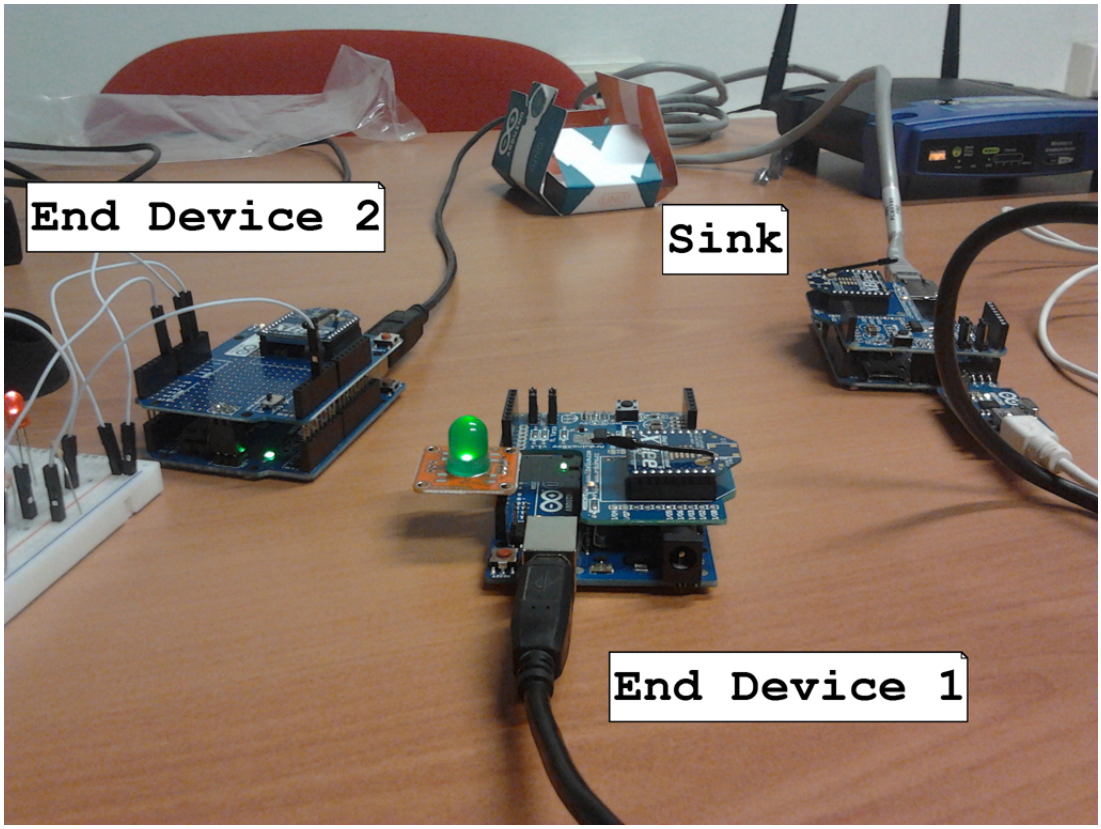



Figura 7.1: Rete a tre dispositivi: Sink e due End Device

Per un esempio di interazione di questi dispositivi con l'applicazione context-aware si veda il capitolo 8.

# Capitolo 8

# Applicazione mobile

La rete di sensori produce una quantità di informazioni che vengono collezionate, raggruppate per tipo ed emesse in broadcast su un'unica interfaccia di rete.
Un'applicazione che sia in grado di leggere il formato di questi messaggi e determinarne la semantica è definita come context-aware, in quanto in grado di elaborare dati estrapolati dal contesto circostante (dati ambientali) ed eventualmente di reagire con specifiche azioni, in maniera del tutto autonoma.

Il seguente capitolo descrive specifica, struttura e implementazione dell'applicazione che risponde a queste caratteristiche.
Sarà descritta l'architettura dell'applicazione, il funzionamento specifico di tutti i suoi moduli e, in maggiore dettaglio, il protocollo utilizzato per comunicare con la rete di sensori.
Data la considerevole differenza nella potenza di calcolo e memoria disponibile della piattaforma di esecuzione, la maggior parte della logica del sistema è racchiusa in questo lato della comunicazione.

Segue una breve descrizione del comportamento dell'applicazione e funzionalità offerte all'utente.

All'avvio e per tutta la durata della sua esecuzione, l'applicazione interagisce con l'interfaccia wifi, rilevando l'eventuale connessione ad un Access Point e notificando di conseguenza l'utente con il SSID della rete.

Stabilita una connessione l'applicazione si pone in ascolto di messaggi provenienti da un'eventuale rete di sensori, scartando quelli che non rispettano il formato adottato.

I messaggi ricevuti sono decodificati e visualizzati con un'interfaccia a lista espandibile, mostrando per ognuno dei dati che li compongono una rappresentazione testuale dipendente dalla lingua di sistema.
Tra i dati più comuni possiamo distinguere valore, collocazione e tipo; in base a quest'ultimo più messaggi possono essere aggregati in un unico elemento nella lista, espanso nelle singole rilevazioni al touch dell'utente.
L'elemento aggregato indica, per una consultazione semplificata, il numero di elementi semplici contenuti in esso e la media dei loro valori, se di tipo numerico.

L'utente può inoltre predisporre l'applicazione a reagire a determinati tipi di eventi: il superamento di soglie minime o massime per il valore numerico di certi dati, quali temperatura e illuminamento, o la ricezione di un valore specifico (per esempio lo stato "On" di uno switch).
Questi eventi possono essere discriminati tra loro in base al tipo di dato e alla collocazione.
Le reazioni potranno essere di due tipi: il lancio di un'applicazione esterna, selezionabile fra quelle installate nel sistema, o la semplice notifica all'utente del valore ottenuto.
I profili che descrivono le coppie evento-reazione sono memorizzati nello storage dell'applicazione, garantendone il recupero al successivo avvio; questi possono essere modificati o cancellati in qualsiasi momento.

L'applicazione svolgerà queste azioni senza necessità di intervento da parte dell'utente, tentando il recupero della connessione nel caso dovesse interrompersi.

## 8.1 Protocollo

Come accennato in precedenza, il formato dei messaggi che i dispositivi Arduino si scambiano tra loro è estremamente conciso, nonché estraneo ad un qualunque linguaggio naturale.
L'applicazione Android, fornendo un'interfaccia alle informazioni ricevute, ha il compito di tradurre le stringhe usate da questo protocollo in una rappresentazione human-readable, preservandone la semantica intesa.
Allo scopo di semplificare l'operazione, nonché rendere il linguaggio utilizzato il più scalabile possibile, viene fatto uso di due diversi files XML integrati all'interno dell'applicazione.

Il primo di questi ha scopo di definire il protocollo utilizzato, cioè i termini che lo compongono, in modo che l'applicazione possa effettuare un semplice parsing dei messaggi, al fine di distinguerne uno valido da uno non valido.

```
<?xml version="1.0" encoding="utf-8"?>
<root
        sn=""
        x0=""
        x1="" >
    <a>
        <a-0 x2="" />
        <a-1
                x2=""
                x3="" />
        <a-2 />
        <a-3 />
    </a>
    <b>
        <b-0 />
    </b>
</root>
```

È implicito che i sotto-termini ereditino gli attributi dal termine padre, al fine di ottimizzare lo spazio utilizzato ed evitare inutili ridondanze (certi attributi saranno comuni a tutti i sensori).
Il secondo file XML rappresenta una sorta di **Meta-livello**, atto a descrivere la semantica intesa per la grammatica precedentemente fornita:

```
<?xml version="1.0" encoding="utf-8"?>
<root
    id="root"
    x0="Value"
    x1="Location" >
    <a id="Sensor" >
        <a-0
            id="Temperature"
            x2="Scale" />
        <a-1
            id="Light"
            x2="Unit"
            x3="Minimum" />
        <a-2 id="Button" />
        <a-3 id="Humidity" />
    </a>
    <b id="Actuator" >
        <b-0 id="LED" />
    </b>
</root>
```

Per ogni tag, quindi ogni oggetto o tipo di informazione, viene introdotto un attributo id, che ne descrive la rappresentazione in un linguaggio naturale.
Il valore assegnato agli attributi, assente nella precedente versione dell'XML, determina allo stesso modo la loro interpretazione.
L'utilizzo di questo meta-livello consente di fornire diverse rappresentazioni interscambiabili (per esempio in base alla lingua scelta); l'applicazione stessa contiene due diversi XML che descrivono il Meta-livello: uno in lingua inglese, l'altro in lingua italiana, automaticamente selezionati in base alla lingua di sistema.

È bene precisare che l'applicazione utilizza il primo dei due per processare le informazioni; il Meta-livello ha solo una funzione descrittiva degli oggetti presi in considerazione.
Ne segue che se il Meta-livello manca di definizioni, o è del tutto assente, l'applicazione mostrerà all'utente i termini usati dai microcontrollori (quindi a-0 al posto di Temperature), senza che il funzionamento ne sia pregiudicato.
Qualora si rivelasse necessario estendere l'insieme di oggetti rappresentati dal protocollo, sarebbe sufficiente aggiungere nuovi elementi alla gerarchia (ed eventualmente al Meta-livello), per esempio:

```
        ...
<a-2 id="Button">
        <a-2-0 id="Power_Button"/>
        <a-2-1 id="Reset_Button"/>
        <a-2-2 id="Panic_Button"/>
</a-2>
       ....
```

Sono quindi introdotti tre diversi tipi di Button: per indicare l'avvio di un macchinario, il suo reset o il verificarsi di uno stato d'allarme.

Infine, la notazione usata per descrivere gli elementi (quindi i tags XML) ha il vantaggio di essere stringata e di facile parsing da parte dell'applicazione; infatti, dato un nodo arbitrario, è semplice risalire ai suoi avi leggendone il solo identificativo (da a-2-2 ad a-2 ad a, in sequenza: Panic Button, Button, Sensor).

## 8.2 Architettura

La struttura dell'applicazione Android, quindi le classi che la compongono e le loro interazioni, è descritta dal seguente diagramma di classe UML:

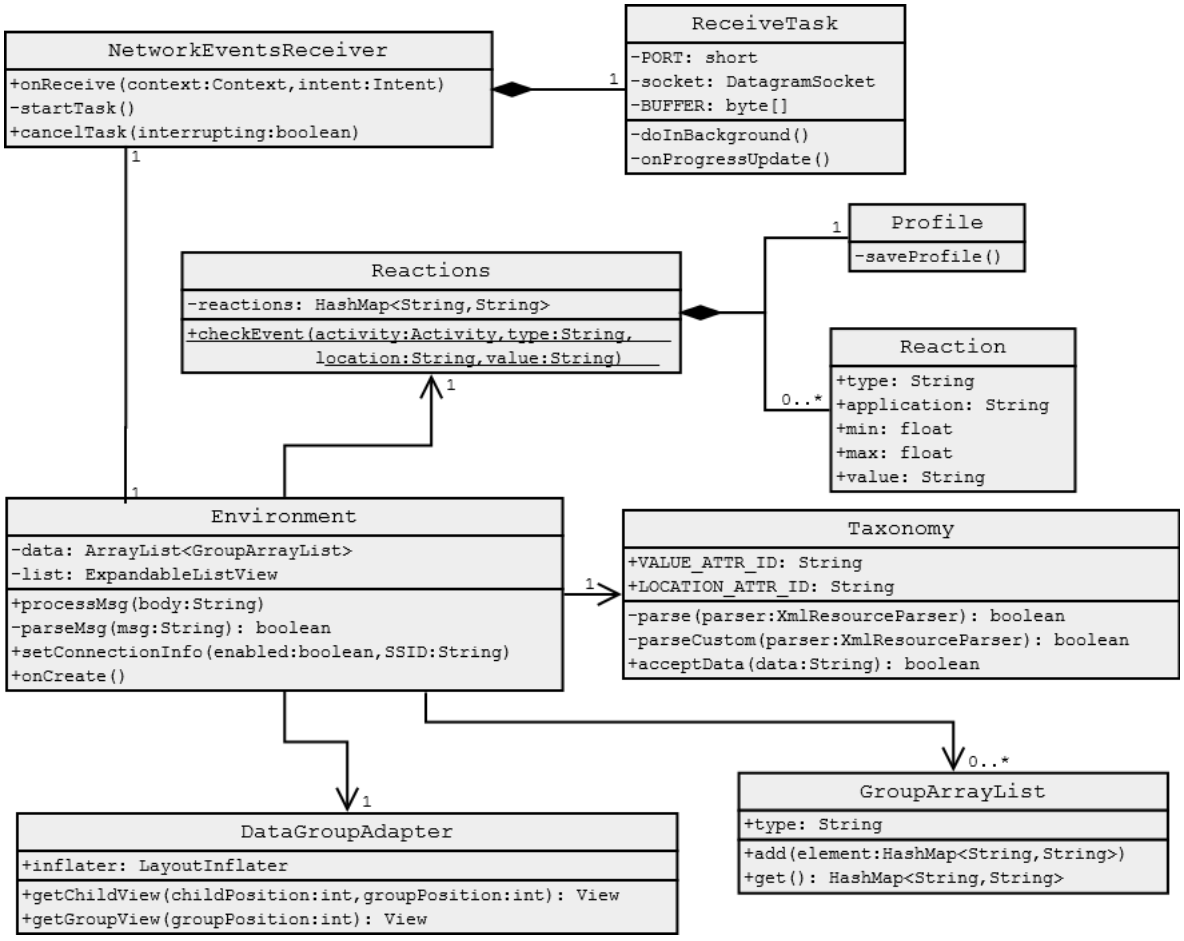



Figura 8.1: Class Diagram dell'applicazione

### 8.2.1 Environment

Questa classe costituisce l'Activity principale dell'applicazione, nonché il punto di accesso a tutti i moduli che la compongono.
Alla creazione (nel metodo onCreate) dell'Activity la classe si occupa di caricare il layout grafico, registrare il NetworkEventsReceiver per gli eventi desiderati, inizializzare la classe Taxonomy con gli identificativi dei file XML utilizzati per descrivere il protocollo (si veda 8.1) e memorizzare gli estremi delle applicazioni installate nell'array usato dalla classe Profile.

I dati collezionati dall'applicazione durante la sua esecuzione sono mantenuti in una lista di GroupArrayList, una classe definita come lista di HashMap (struttura dati a dizionario) con chiavi e valori di tipo String.
Alla ricezione di un pacchetto UDP sull'interfaccia di rete il metodo processMsg di questa classe ne controlla il payload, riconoscendo l'eventuale presenza di messaggi multipli e controllando per ognuno di loro la correttezza sintattica all'interno del linguaggio definito dal protocollo utilizzato, mediante la classe Taxonomy.
Per ogni messaggio correttamente riconosciuto il metodo controlla la presenza di vecchie versioni nella lista utilizzata, determinando in base al Sequence Number incluso (qualora fosse presente) se scartare o no il vecchio messaggio, provocando un aggiornamento della rappresentazione grafica su schermo.

La lista è condivisa con la classe DataGroupAdapter, in modo che ogni aggiornamento dei dati posseduti si rifletta di conseguenza sulla loro rappresentazione grafica, mediante invocazione del metodo notifyDataSetChanged della stessa.

È opportuno menzionare che le operazioni sulle strutture dati utilizzate hanno al più un costo lineare nel numero di tipi di dati diversi che l'applicazione ha ricevuto.
Infatti la maggior parte delle operazioni avviene su strutture dati di tipo HashMap, quindi con un costo di ricerca costante.

I dati sono rappresentati graficamente mediante una ExpandableListView, onde consentirne una intuitiva visualizzazione per gruppi, qualora siano dello stesso tipo.
Fra le informazioni aggiunte dall'applicazione è compresa un'indicazione dell'istante di ricezione di un particolare dato.

La classe si occupa anche di mostrare all'utente la presenza di una connessione wifi, visualizzando in tal caso il SSID trasmesso dall'Access Point.

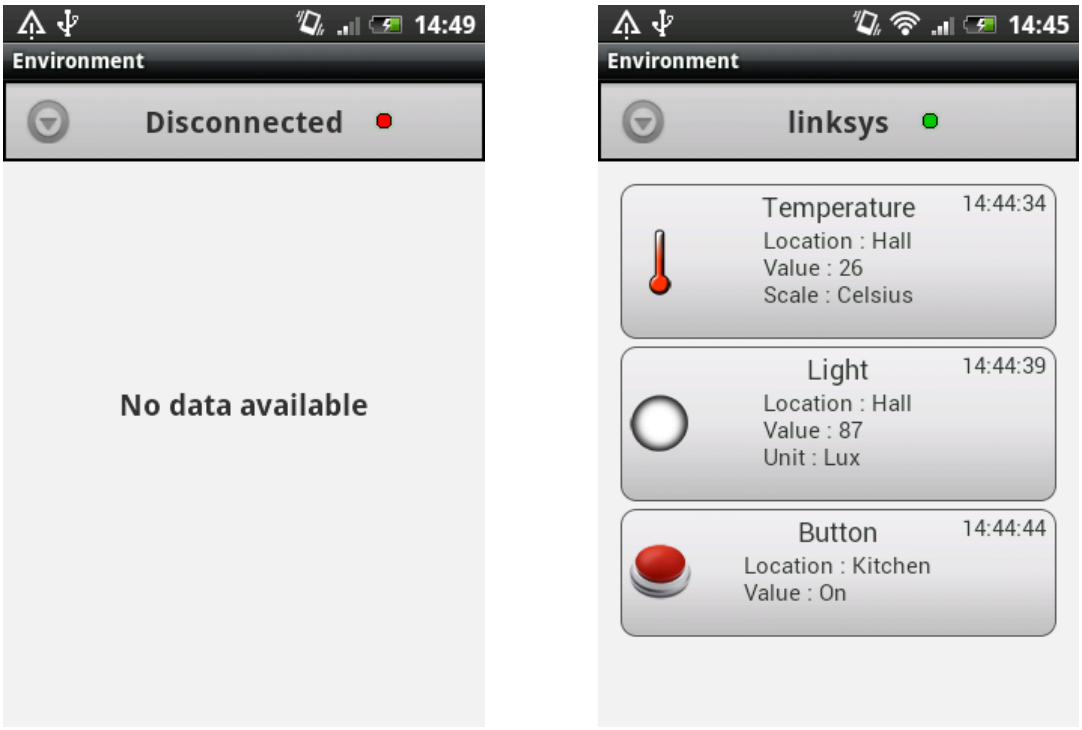
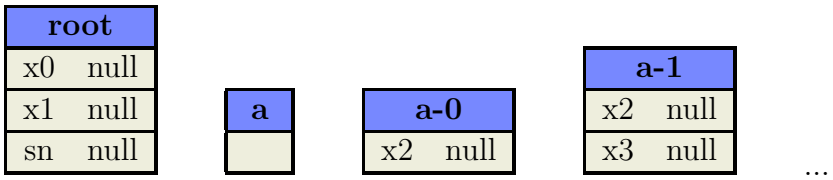



Figura 8.2: Interfaccia grafica dell'applicazione

### 8.2.2 Taxonomy

La classe Taxonomy incapsula al proprio interno l'utilizzo della gerarchia definita dal protocollo e dell'eventuale Meta-livello che ne descrive il significato.
Per evitare ripetuti accessi a memoria secondaria la classe memorizza al suo interno tutte le definizioni di oggetti e attributi della gerarchia, ereditando da HashMap e mappando chiavi di tipo String (nome oggetto) in valori di tipo HashMap da String a String (ogni attributo nella sua rappresentazione human-readable). Tra questi attributi è compreso quello con chiave id, che come valore assume il nome descrittivo dell'oggetto.

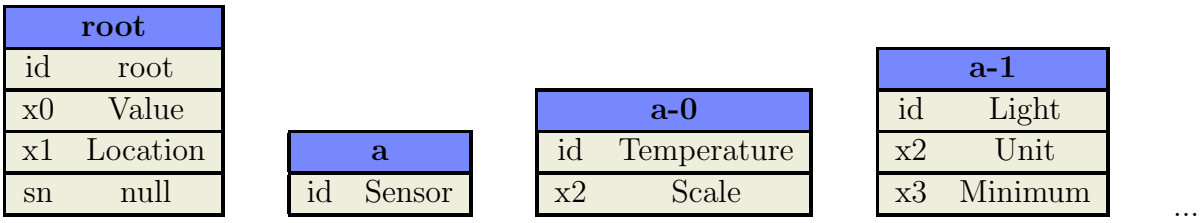
A inizializzazione viene effettuato il parsing del file XML che descrive la gerarchia, memorizzandone tutti i nodi come chiavi nella HashMap e i nomi degli attributi all'interno della relativa HashMap annidata. Si ottiene un risultato di questo tipo:




Questa prima fase ha scopo di fissare i termini accettati all'interno del protocollo, lasciando vuoti i valori relativi ad ogni attributo.
Si noti che la struttura a dizionario così costruita non conserva la relazione gerarchica fra i vari elementi; di fatto non ne ha necessità, poiché in fase di parsing di un messaggio si potrà semplicemente scomporre la stringa che ne rappresenta il tipo, al fine di risalire ai suoi avi.

La fase successiva dell'inizializzazione consiste nel parsing del file XML che descrive il meta-livello: per ogni corrispondenza fra un tag ed una chiave all'interno della HashMap precedentemente costruita, la relativa traduzione viene memorizzata all'interno della HashMap annidata.

Il risultato ottenuto è il seguente:




La voce id viene aggiunta a indicare la traduzione del tipo a opera del meta-livello e ogni attributo, quindi chiave all'interno della HashMap annidata, viene trattato allo stesso modo. In caso di traduzioni mancanti il valore assegnato ad un attributo viene semplicemente lasciato vuoto.

La classe è anche responsabile del parsing di un singolo messaggio pervenuto dalla rete tramite la funzione acceptData: dopo averlo convertito in un oggetto JSON, si cerca il valore della chiave id per identificare il tipo di dato e, successivamente, tutti i suoi attributi all'interno della relativa HashMap annidata, costruita in fase di inizializzazione. Se un attributo non fosse presente nella HashMap lo si cercherebbe fra quelli del nodo genitore, continuando fino ad arrivare al tag radice (root).
Se la ricerca di un attributo non produce risultati, o non esiste una chiave uguale al nome del tipo, il messaggio viene semplicemente rifiutato; altrimenti la funzione restituisce una HashMap che mappa tutte le chiavi del messaggio (ognuna sostituita con la sua traduzione human-readable, se presente) nei rispettivi valori.
Per chiarire il risultato, si consideri il seguente esempio:

La stringa

{id : a-2, x0 : Off, x1 : Room1}

Viene tradotta dalla funzione in

{id : Button, Value : Off, Location : Room1}

### 8.2.3 NetworkEventsReceiver

La classe NetworkEventsReceiver ha il compito di gestire l'interfacciamento con la rete wifi, ponendosi in attesa di eventi relativi alla connessione o disconnessione da un Access Point e ricevendo pacchetti UDP sulla porta specificata.

La classe eredita da BroadcastReceiver, ponendosi in attesa degli eventi specificati dalla costante CONNECTIVITY_ACTION, inerenti la connessione e disconnessione da una rete generica.
Combinando la ricezione di questi eventi con un controllo sull'attuale stato dell'antenna wifi, tramite WifiManager, è possibile mettere il processo in ascolto sulla porta UDP designata solo al momento della effettiva connessione ad un Access Point.

La ricezione di pacchetti avviene su un Thread separato da quello principale, al fine di evitare il blocco dell'intera applicazione, creato mediante la classe ReceiveTask, definita come sottoclasse di AsyncTask.
Questa classe è in grado di incapsulare al suo interno la comunicazione col Thread UI, passando il payload dei pacchetti ricevuti al metodo processMsg della classe Environment.

Qualora la connessione dovesse cadere, l'antenna wifi essere disabilitata o semplicemente l'utente chiudesse l'applicazione, la ReceiveTask sarebbe cancellata allo scadere del successivo timeout sulla connessione.

### 8.2.4 DataGroupAdapter

Questa classe eredita da BaseExpandableListAdapter e fornisce una fondamentale mediazione tra i dati collezionati e l'interfaccia grafica su cui sono rappresentati, aggiornando la ExpandableListView che li descrive.
Infatti previo aggiornamento della lista di dati memorizzati è sufficiente invocare il metodo notifyDataSetChanged di questo adapter per avere un aggiornamento su schermo.
La classe è, in sostanza, responsabile di rappresentare i soli attributi di qualche utilità (non gli eventuali Sequence Numbers relativi ai messaggi, per esempio) e di aggregare in un unico oggetto espandibile i dati dello stesso tipo.
Se il tipo di dato ha valore numerico (la temperatura indicata, per esempio), nell'aggregazione la classe calcola la media aritmetica fra i valori rilevati, mostrandola all'utente.

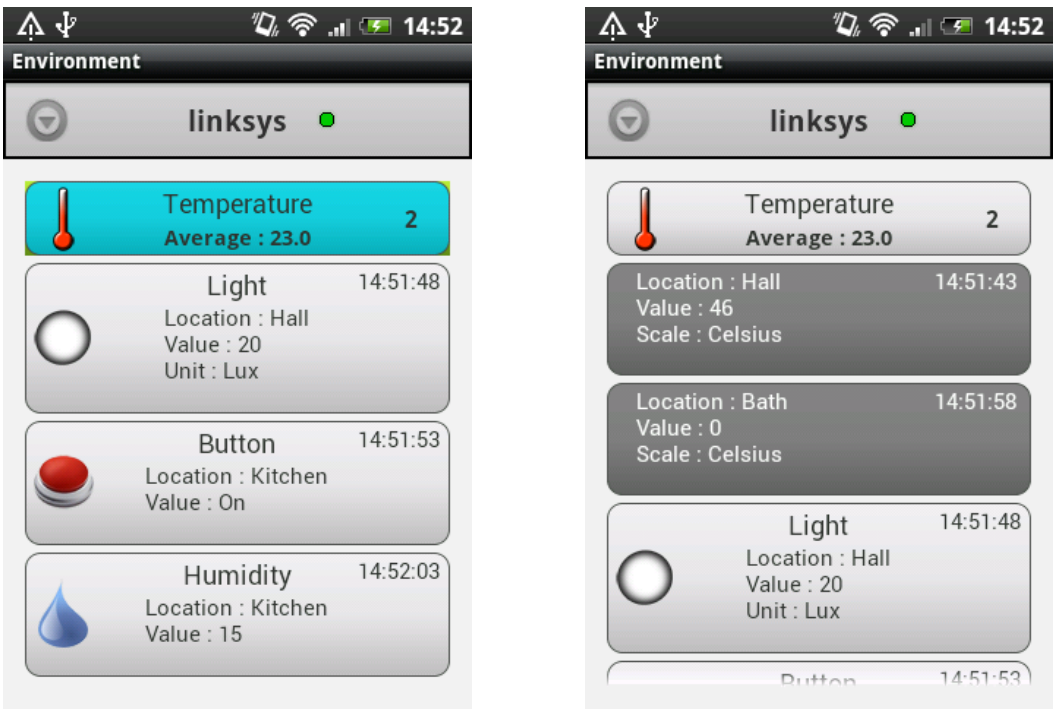



Figura 8.3: Lista di valori aggregati

### 8.2.5 Reactions

La classe Reactions incapsula tutta la logica relativa alla componente "esecutiva" dell'applicazione, ossia le azioni da intraprendere al verificarsi di determinati eventi.
Per meglio esplicare i compiti svolti da questo modulo si descrivono le due classi ausiliarie di cui fa uso: Reaction e Profile.

La classe Reaction rappresenta la reazione ad un evento, descrivendo con le sue variabili il tipo di intervento (applicazione o notifica), i valori esatti, minimi o massimi aspettati e l'eventuale applicazione da eseguire.
Diversi oggetti di tipo Reaction sono memorizzati come valori in una HashMap, le cui chiavi consistono in coppie di stringhe tipo-collocazione.

La classe Reactions, ereditando da ListActivity, consente una visualizzazione a lista dei valori nella HashMap:

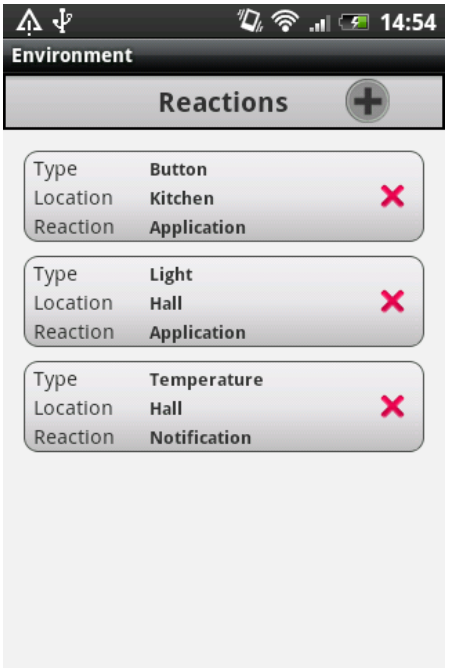



Figura 8.4: Lista di profili evento-reazione

Alla ricezione di un nuovo messaggio, dopo aver controllato tramite Sequence Number che non sia un duplicato, si verifica nella funzione checkEvent che non corrisponda ad un determinato evento.
Il controllo si effettua, in sequenza, sui campi value, min e max dell'oggetto Reaction, verificando che il valore non coincida col primo, non sia inferiore al secondo o superiore al terzo. Se una delle condizioni non è rispettata viene emessa una notifica o eseguita l'applicazione scelta, a seconda del valore del campo type dell'oggetto Reaction.
L'identificativo numerico assegnato a ogni oggetto di questa classe ha lo scopo di distinguere le notifiche emesse, in modo da sovrascrivere solo quelle relative allo stesso evento.
Gli oggetti di tipo Reaction sono memorizzati sullo storage dell'applicazione mediante il meccanismo delle **Shared Preferences**, usando come chiavi le stesse della HashMap. Tramite conservazione delle diverse chiavi in un ulteriore oggetto di tipo Shared Preferences (il cui identificativo è noto) è inoltre possibile recuperarle tutte all'avvio dell'applicazione.

La classe Profile eredita da Activity e fornisce all'utente l'interfaccia grafica necessaria alla creazione di nuovi profili:

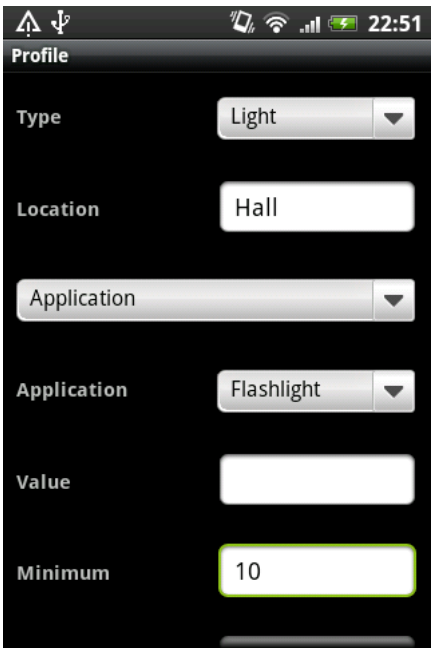



Figura 8.5: Interfaccia di impostazione dei profili

Per facilitare la configurazione dei diversi profili, le stringhe che descrivono il tipo dei dati rilevati vengono memorizzate in un array dalla classe Taxonomy, in fase di parsing; questo array viene successivamente utilizzato da uno Spinner per fornire i dati all'utente. Stessa cosa, nel metodo onCreate della classe Environment, avviene per le applicazioni installate sul sistema.

Il risultato è il seguente:

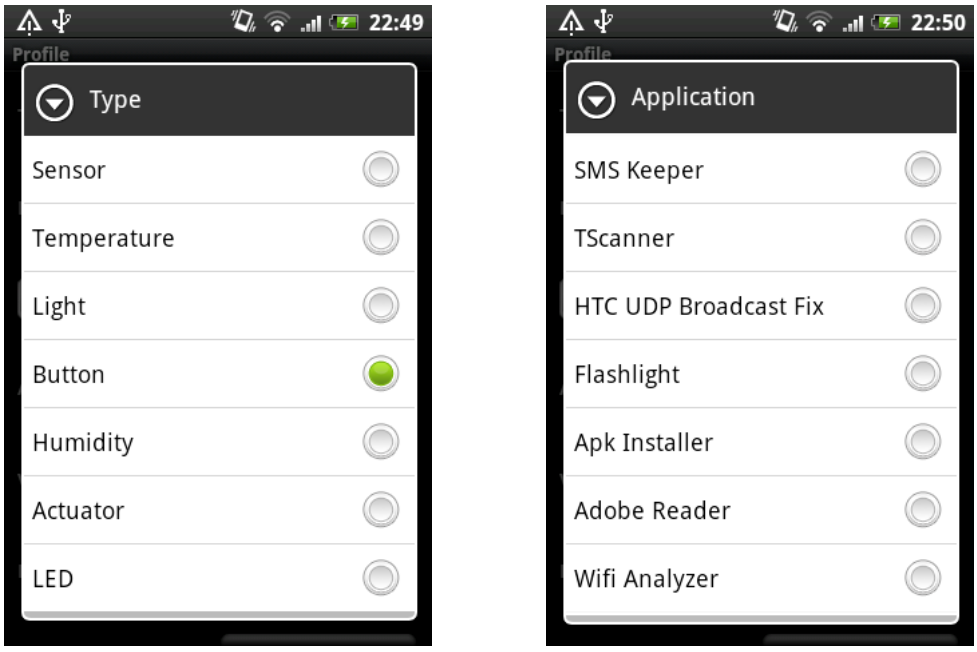



Figura 8.6: Liste di dati rilevati e applicazioni installate

Alla chiusura dell'Activity viene chiamato il metodo saveProfile, responsabile della lettura dei valori immessi nei vari campi e conseguente memorizzazione nella HashMap usata dalla classe Reactions.

Le seguenti foto mostrano l'interazione dell'applicazione con la rete rappresentata nella figura 7.1.
In particolare il valore di illuminamento, prodotto dal fotoresistore connesso ad Arduino tramite breadboard, ha valore nullo a luce spenta (nota bene: per la semplice dimostrazione non si è convertito il valore analogico in Lux, sebbene il microcontrollore invii anche tale indicazione).

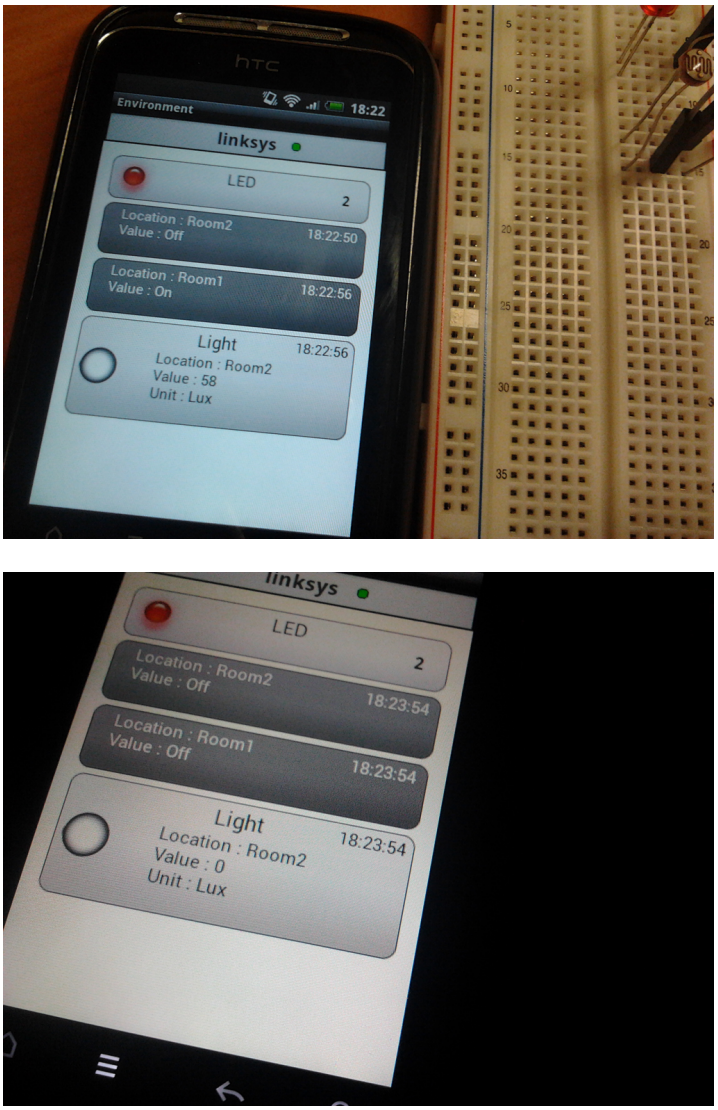



Figura 8.7: Interazione tra applicazione e rete di sensori

## 8.3 Sviluppi

A partire da queste specifiche vi sono numerose evoluzioni possibili per l'applicazione, tra le quali:

- Rendere più flessibile il sistema di profili basati su eventi e reazioni. L'utente dovrebbe poter raggruppare più casistiche nello stesso profilo, disgiunte o congiunte fra loro, fissando un ordine basato su priorità.
- Inserire la possibilità di filtrare i dati ottenuti, per tipo o collocazione.
- Pubblicare i files XML che descrivono il protocollo di comunicazione ed eventuali Meta-livelli sul web, inserendo un sistema di aggiornamento automatico, tramite Download Manager, nell'applicazione.
- Fornire un punto di accesso globale ai dati rilevati per le altre applicazioni presenti nel sistema, usando un Content Provider unico o vari distinti per tipo di dato.
- Rendere la comunicazione con la rete bidirezionale, permettendo di lanciare comandi specifici quali attivazione di attuatori o controllo diretto di specifici sensori.
- Inserire la possibilità di condividere i dati rilevati con altri utenti, via Internet, o pubblicarli su siti web tramite richieste POST in un formato predefinito.

# Conclusioni

L'obiettivo di questo progetto era quello di costruire una rete di sensori che dialogasse con un'applicazione Context Aware, in esecuzione su un dispositivo mobile.

Per costruire la rete si è fatto uso di schede Arduino connesse tra loro mediante modem modello XBee, utilizzando i protocolli ZigBee per comunicazioni wireless.
Tramite l'utilizzo di un Arduino Ethernet è stato possibile integrare due diverse interfacce di rete sulla stessa scheda e fornire una mediazione tra la rete ZigBee ed un'altra esterna.
L'applicazione mobile è stata sviluppata su piattaforma Android.

La costruzione della rete ha portato alla luce alcune problematiche, tra le quali la scarsa disponibilità di memoria dei dispositivi utilizzati.
Infatti ci si è resi conto che per manipolare ingenti quantità di informazioni, in una rete più vasta, una topologia a stella in cui il nodo centrale (il Sink) gestisca tutti i dati prodotti non è una soluzione realistica. Occorrerebbe in questo caso decentralizzare la raccolta di informazioni e costruire una rete formata da numerosi Cluster, ognuno dei quali emittente un broadcasting di informazioni differenziate (probabilmente per località).

D'altra parte i protocolli ZigBee si sono rivelati decisamente i più adeguati alle esigenze imposte da una rete costituita da dispositivi a bassa potenza di calcolo; l'utilizzo di una modalità di comunicazione trasparente, tra il microcontrollore e il modem, che permettesse di ridurre l'overhead derivante dalla composizione di pacchetti con un formato specifico, ha costituito la scelta più vantaggiosa.

Il protocollo utilizzato per le comunicazioni tra la rete e le applicazioni che interagiscono con essa viene descritto con una struttura gerarchica di tipi di dato, in formato XML; questo presenta un duplice vantaggio: una maggiore scalabilità, consentendo un facile ampliamento dei termini utilizzati, e la possibilità per applicazioni di natura eterogenea e in esecuzione su piattaforme diverse di comunicare con la rete.
Poiché le informazioni vengono emesse in broadcast sulla rete a cui è connesso il Sink, diverse applicazioni possono riceverle in simultanea e processarle in modo differente.

Lo sviluppo più naturale per un tale sistema è quello di rendere la comunicazione tra i due estremi bidirezionale: permettere quindi ad un dispositivo smartphone di inviare messaggi alla rete, attivando per esempio un attuatore remotamente.
Tale evoluzione risulta sicuramente fattibile: il dispositivo mobile estrae l'indirizzo IP del nodo Sink dal pacchetto UDP ricevuto ed invia ad esso un messaggio, specificando in esso il nodo selezionato e l'azione richiesta.
L'identificazione univoca del nodo tramite la stringa che descrive la collocazione potrebbe rivelarsi sufficiente a questo scopo.
Il Sink invierebbe successivamente un pacchetto in broadcast sulla rete ZigBee, ignorato dai nodi che non corrispondano al destinatario, o semplicemente, utilizzando la modalità API dei moduli XBee, direttamente al nodo interessato.
Le problematiche non sono tuttavia banali: sebbene questo meccanismo non presenti un aumento eccessivo di complessità per il Sink, i nodi End Device sarebbero costretti a porsi in ricezione con cadenza regolare, riducendo drasticamente i loro periodi di sleep.
Con l'aumento delle dimensioni della rete, diventerebbe inaccettabile per eventuali nodi padre (Routers o lo stesso Coordinator) mantenere in buffer i pacchetti destinati ad un nodo figlio.
Inoltre, ridurre drasticamente i tempi di sleep, per dispositivi che necessitano di lunghi periodi di autonomia, rappresenta un trade-off decisamente non trascurabile.

Un'altra potenziale evoluzione consiste nella mobilità del nodo Sink.
Se il nodo Sink fosse capace di muoversi, autonomamente in base a certi algoritmi o radiocomandato dall'utente, si potrebbe dividere la rete in più sottoreti indipendenti, riducendone drasticamente la complessità.
Ogni sottorete risultante farebbe capo al medesimo Coordinatore (il Sink), il quale passerebbe dall'una all'altra semplicemente spostandosi nel raggio d'azione dei suoi nodi.
Una tale possibilità sarebbe sicuramente di grande utilità in casi di emergenza, qualora una sezione della rete dovesse cadere e occorresse un immediato cambio di contesto, ma presenta uno svantaggio analogo a quello del caso precedentemente discusso: la mancanza di alimentazione.
Per definizione, un Coordinator dovrebbe essere alimentato elettricamente via cavo. La quantità di elaborazione necessaria per costruire e mantenere la rete ZigBee, nonché la gestione del mezzo di movimento e dell'interfaccia con le applicazioni esterne (wifi o bluetooth che sia), costituirebbero un notevole dispendio di energia per un dispositivo alimentato a batteria.
Questi fattori renderebbero problematico l'utilizzo costante di un Sink mobile.

Analizzando un caso reale di utilizzo, si pensi al controllo e automazione di un edificio.
La rete, la cui portata ottimale supera i 200 metri, emetterebbe informazioni accessibili a tutti i dipendenti provvisti di uno smartphone che esegua l'applicazione.
Tramite essa sarebbero in grado di monitorare lo stato di varie postazioni: per esempio controllare che gli interruttori di un'apparecchiatura siano spenti o accesi, determinare il grado di utilizzo di un macchinario mediante il numero di pressioni del bottone che ne regola la frequenza o la temperatura attuale in luoghi dove essa rappresenta un fattore critico.
Simultaneamente un computer interfacciato sulla stessa rete potrebbe ricevere gli stessi dati (emessi in broadcast, si ricorda) e inoltrarli via Internet a elaboratori situati in diverse strutture, per confronto e aggregazione.
Inoltre i dipendenti potrebbero essere in grado di attivare remotamente degli attuatori, il cui stato sarebbe immediatamente notificato a tutti gli altri nello stesso edificio.

Per concludere possiamo affermare che sicuramente, dati gli sviluppi futuri che i paradigmi di Internet of Things e Context-Aware Computing promettono, un'integrazione dei due nel medesimo sistema non potrebbe che portare grandi migliorie in qualunque settore.

# Appendice A

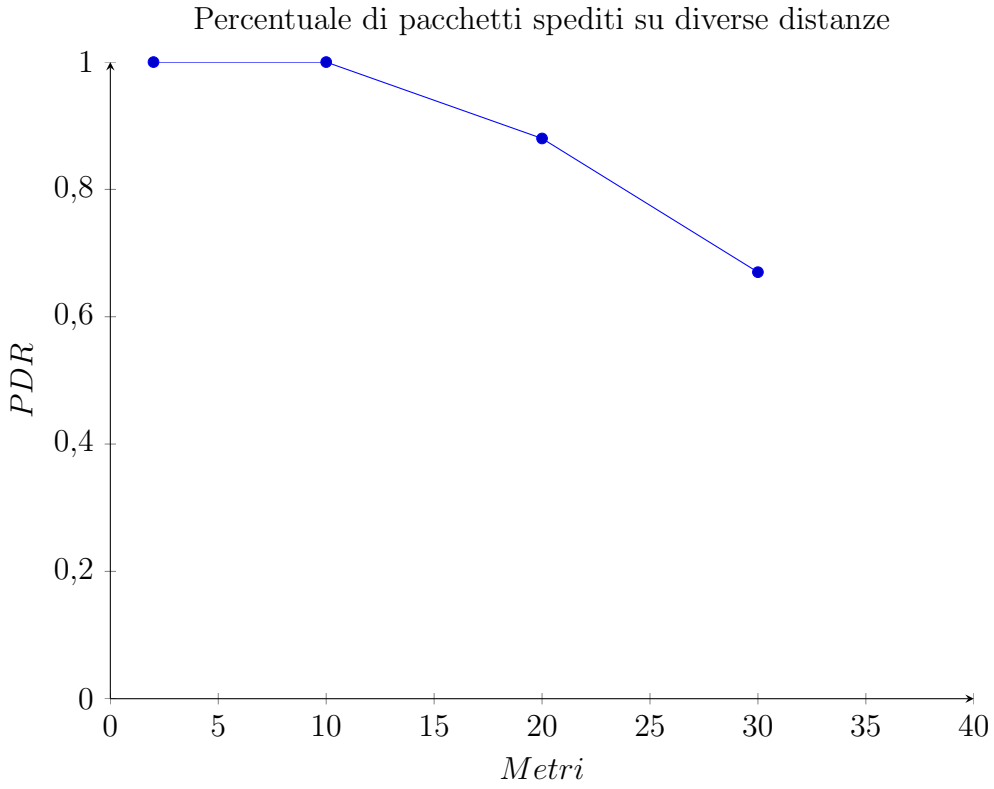
# Analisi di prestazioni

In questa appendice saranno esposti dei risultati ottenuti su test di comunicazione diretta fra due moduli XBee.
Lo scopo dei test è quello di valutare la qualità della comunicazione fra due moduli in termini di Packet Delivery Ratio (PDR), calcolata come $\frac{\#\,Pacchetti\,ricevuti}{\#\,Pacchetti\,inviati}$.

Per le rilevazioni si è utilizzata la funzionalità di Range Test del tool X-CTU.
Dal nodo monitorato da X-CTU, configurato come End Device, si sono spediti 100 pacchetti contenenti un payload di 32 bytes.
Il Coordinator utilizzato come destinatario è stato collegato ad un Arduino Uno tramite XBee Shield e programmato come loopback, quindi per rispedire indietro tutti i pacchetti ricevuti dall'End Device.
La stima si è quindi effettuata in base al numero di risposte ricevute da quest'ultimo, applicando un time-out di 200 millisecondi su ogni ricezione.
Il campo RSSI (Received Signal Strength Indication) rappresenta, in una scala da -100 a -40, la potenza del segnale ricevuto dall'antenna del modulo XBee.

| Distanza (metri) | RSSI medio | Pacchetti ricevuti | PDR |
|---|---|---|---|
| 2 | -44 | 100 | 1 |
| 10 | -65 | 100 | 1 |
| 20 | -92 | 88 | 0.88 |
| 30 | -95 | 67 | 0.67 |

Si vuole precisare che le trasmissioni sono state effettuate in interni e che le distanze riportate non sono esatte, ma delle approssimazioni.
Il seguente diagramma rappresenta la relazione che intercorre nella distanza fra i due nodi e la Packet Delivery Ratio registrata.




Dal grafico si può notare come, entro una certa distanza, il segnale sia sufficientemente potente da permettere la ricezione del 100 % dei pacchetti.

Va considerato che in questi test X-CTU non pone attese fra l'invio dei singoli messaggi, se non la ricezione della risposta al precedente o lo scadere del timeout di 200 millisecondi. Da questo si possono giudicare positivi i risultati ottenuti, se valutati nell'ambito di una rete di sensori che prevede uno scarso traffico di messaggi.

# Bibliografia


[1] Android Developers, http://developer.android.com

[2] Arduino Playground, http://www.arduino.cc/playground/

[3] Banzi Massimo (2012), *Arduino. La guida ufficiale*, Tecniche Nuove

[4] Bernardo Giovanni (2011), *Easy Bee*, Rev. 1,

[5] Digi International Inc. (2010), *X-Bee s2 Datasheet*, http://www.digi.com

[6] Gargenta Marko (2011), *Learning Android*, 1st Edition, Sebastopol, O'Reilly.

[7] ZigBee Alliance (2006), *ZigBee Specifications*, versione 1.0 r13, http://www.zigbee.org/

[8] ZigBee Network (2012), *Il Protocollo IEEE 802.15.4 e ZigBee*, http://www.zigbeenetwork.it/

# Elenco delle figure

# Ringraziamenti

Desidero ringraziare Luciano Bononi, Marco Di Felice e Luca Bedogni, per avermi fornito tutta l'attrezzatura necessaria allo svolgimento della mia tesi, spazio nel quale operare in assoluta libertà e supporto e assistenza continue.

Ringrazio la mia famiglia per avermi sostenuto e appoggiato in questi anni e, più che mai, adesso.

Ringrazio Eugenia per essermi stata vicina ed essersi interessata tanto al mio lavoro.

Un ultimo ringraziamento ai miei compagni di studio, che sono stati soprattutto degli amici, per i loro incoraggiamenti e l'aiuto nello svolgimento dei test integrati in questo documento.