ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

MIDDLEWARE PER INTERNET OF THINGS: JAVA EMBEDDED COME CASO DI STUDIO

Elaborata nel corso di: Sistemi Operativi

Tesi di Laurea di:
GIANLUCA SPADAZZI

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2013–2014
SESSIONE II

PAROLE CHIAVE

Internet of Things

Wireless Sensor Network

RFID

Middleware

Java Embedded

Alla mia famiglia, alla mia ragazza e ai miei amici

Indice

Introduzione

Grazie al progresso dell'elettronica, ai giorni nostri è possibile costruire dispositivi elettronici molto piccoli, che col passare del tempo lo sono sempre più. Questo ci permette di poter imboccare nuove strade nel mondo dell'informatica, sfruttando proprio questo fatto. Le dimensioni ridotte dei dispositivi in commercio, come sensori, attuatori, tag e tanto altro, sono particolarmente adatte a nuovi scenari applicativi. Internet of Things è una visione in cui Internet viene esteso alle cose. Facendo largo uso di dispositivi come sensori e tag è possibile realizzare sistemi intelligenti che possono avere riscontri positivi nella vita di tutti i giorni. Tracciare la posizione degli oggetti, monitorare pazienti da remoto, rilevare dati sull'ambiente per realizzare sistemi automatici (ad esempio regolare automaticamente la luce o la temperatura di una stanza) sono solo alcuni esempi. Internet of Things è la naturale evoluzione di Internet, ed è destinato a cambiare radicalmente la nostra vita futura, poichè la tecnologia sarà sempre più parte integrante della nostra vita, aumentando sempre più il nostro benessere e riducendo sempre più il numero delle azioni quotidiane da compiere. Sempre più sono i middleware, le piattaforme e i sistemi operativi che nascono per cercare di eliminare o ridurre le problematiche relative allo sviluppo di sistemi di questo genere, e lo scopo di questa tesi è proprio sottolinearne l'importanza e di analizzare gli aspetti che questi middleware devono affrontare. La tesi è strutturata in questo modo: nel capitolo uno verrà fatta una introduzione a Internet of Things, analizzando alcuni degli innumerevoli scenari applicativi che ne derivano, insieme però alle inevitabili problematiche di tipo tecnologico e sociale. Nel secondo capitolo verranno illustrate le tecnologie abilitanti di Internet of Things, grazie alle quali è possibile realizzare sistemi intelligenti. Nel terzo capitolo verranno analizzati gli aspetti relativi ai middleware, sottolineandone l'importanza e prestando attenzione alle funzioni che devono svolgere, il tutto riportando anche degli esempi di

middleware esistenti. Nel quarto capitolo verrà approfondito il middleware Java Embedded di Oracle.

Capitolo 1

Introduzione a Internet of Things

Internet of Things (IoT) rappresenta una visione in cui Internet si estende dentro il mondo, abbracciando gli oggetti di ogni giorno. Gli oggetti fisici non sono più disconnessi dal mondo virtuale, ma possono essere controllati da remoto e possono essere dei punti di accesso fisici a Internet. Non solo oggetti, ma anche animali o persone, sono marcati con identificatori univoci, e possono inviare dati attraverso la rete. La comunicazione può avvenire tra oggetto e oggetto (Machine to Machine), ma anche con gli esseri umani (Machine to Human). E' un settore di grande interesse e importanza, come testimoniato dai dati. Secondo Inc. Gartner, i prodotti di IoT passeranno da 1.9 bilioni di unità nel 2009, a un numero circa tre volte superiore nel 2020, garantendo anche un forte impatto positivo sull'economia. L'idea alla base di IoT è la presenza pervasiva attorno a noi di una varietà di oggetti (ad esempio RFID (identificatori a radio frequenza), sensori, attuatori, tag, dispositivi mobili), che sono in grado di interagire tra di loro e cooperare per raggiungere un obiettivo comune. Un buon esempio di come funziona l'architettura di IoT può essere trovato facendo un'analogia con l'organizzazione sociale degli individui. Gli individui costituiscono nazioni governate da certe regole. Le nazioni a loro volta, pur essendo diverse, possono comunque comunicare e cooperare tra di loro. IoT non è il risultato dello sviluppo di una singola tecnologia, ma dell'aggregazione di tante. Alcuni degli aspetti principali di questa visione sono:

- *Comunicazione e cooperazione*: gli oggetti comunicano tra loro grazie

alla rete.

- *Identificazione*: gli oggetti, grazie a identificatori univoci, possono essere identificati.
- *Rilevazione*: gli oggetti possono raccogliere dati grazie a sensori.
- *Attuazione*: attuatori contenuti negli oggetti, sulla base dei dati raccolti dai sensori, possono manipolare l'ambiente.
- *Elaborazione delle informazioni integrata*: gli oggetti incorporano un microprocessore, o un microcontrollore, e della memoria. Questo serve per memorizzare i dati raccolti e elaborarli.
- *Localizzazione*: grazie al GPS e alla rete, segnali a banda ultra larga etc, gli oggetti sono consapevoli della propria posizione.
- *Interfaccia utente*: gli oggetti comunicano con gli utenti sia direttamente sia indirettamente (ad esempio grazie a uno smartphone, elemento sempre più centrale, come mostra la figura 1.1). L'interazione con l'utente può avvenire in diversi modi, come ad esempio display, gesture, voce etc.

Prima di andare nello specifico e trattare aspetti tecnologici, conviene parlare brevemente della storia di IoT e soffermarsi sugli scenari applicativi che ne emergono, per capire i vantaggi che questo scenario può portare, e analizzare le problematiche che ne conseguono, che possono essere di tipo sociale e di tipo tecnologico.

1.1 Breve storia

Il termine Internet of Things è stato coniato nel 1999 dall'inglese Kevin Ashton, ma l'idea che sta dietro a tutto ciò era già presente in passato (già dalla nascita stessa di Internet). Secondo Richard Yonk, IoT è la naturale evoluzione delle telecomunicazioni. I computer diventano sempre più piccoli e più economici (oltre che di più numericamente) col passare del tempo, e se si va avanti in questa direzione, l'unico modo che ci resterà per interagirci sarà attraverso l'ambiente. Di seguito verranno mostrati alcuni dati relativi alla crescita di dispositivi connessi a Internet nel tempo. Nel 2003 sulla Terra

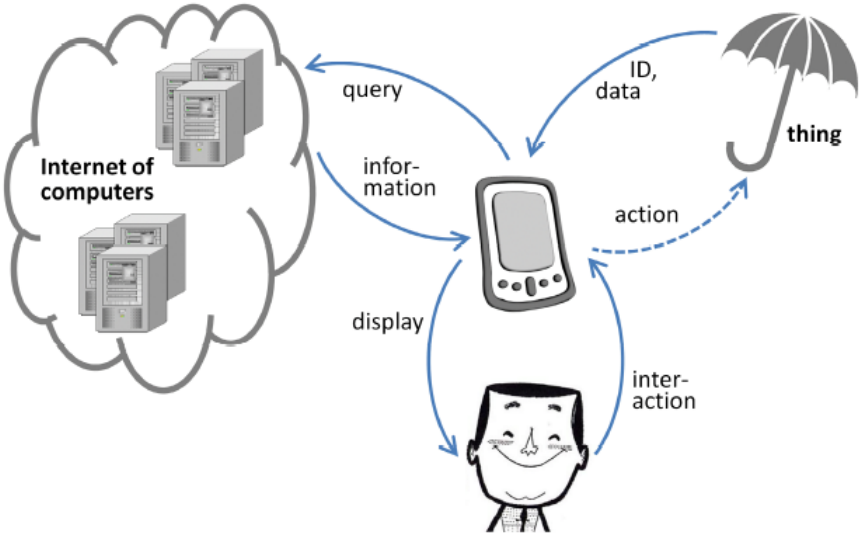

Figura 1.1: Centralità dello smartphone

vivevano all'incirca 6,3 miliardi di persone e i dispositivi connessi a Internet erano più o meno 500 milioni. Dividendo il numero di dispositivi connessi per la popolazione mondiale, il risultato (0,08) è meno di un dispositivo per ogni persona. Nel 2003 l'Internet delle cose si può dire che non esistesse ancora, dal momento che il numero di oggetti connessi era relativamente basso. Dispositivi usabili in qualsiasi luogo come gli smartphone erano stati introdotti da poco. Nel 2010 l'incredibile boom di smartphone, tablet e PC ha portato il numero di dispositivi connessi a Internet a 12,5 miliardi, mentre la popolazione mondiale è salita a 6,8 miliardi. Per la prima volta nella storia, quindi, il numero di dispositivi connessi per persona ha superato quota uno (attestandosi per l'esattezza a 1,84), come mostrato in figura 1.2. Cisco IBSG prevede che i dispositivi connessi a Internet raggiungeranno quota 25 miliardi entro il 2015 e 50 miliardi entro il 2020. I calcoli non tengono conto dell'evoluzione della tecnologia ma solo dei dati attuali, e le stime sono fatte su tutta la popolazione mondiale (considerando quindi anche le aree povere, in cui molte persone non possiedono dispositivi mobili). Inoltre va anche considerato che la connessione a Internet progressivamente si espande anche a oggetti, animali, persone e tanto altro, facendo così

impennare i dati sui dispositivi connessi. Il primo oggetto in grado di connettersi a Internet fu una distributrice di Coca Cola della Carnegie Mellon University, che poteva inviare messaggi contenenti la disponibilità o meno di lattine, in modo da evitare inutili viaggi verso la zona merenda. Erano però esperimenti soprattutto amatoriali, poichè il vero interesse da parte di aziende e di consumatori nasce tra il 1900 e il 2000. In questo periodo Ashton ricercava insieme all'Auto-ID Center nel campo dell'identificatione radio-frequenza. In seguito, nel 1944, verrà creato da Intel, Cisco, GE e IBM l'Industrial Internet Consortium, nato per incrementare gli standard per l'interoperabilità tra diversi dispositivi. Ad oggi, la presenza di dispositivi di tipo diverso che devono comunicare tra di loro, rappresenta uno dei maggiori problemi da affrontare.

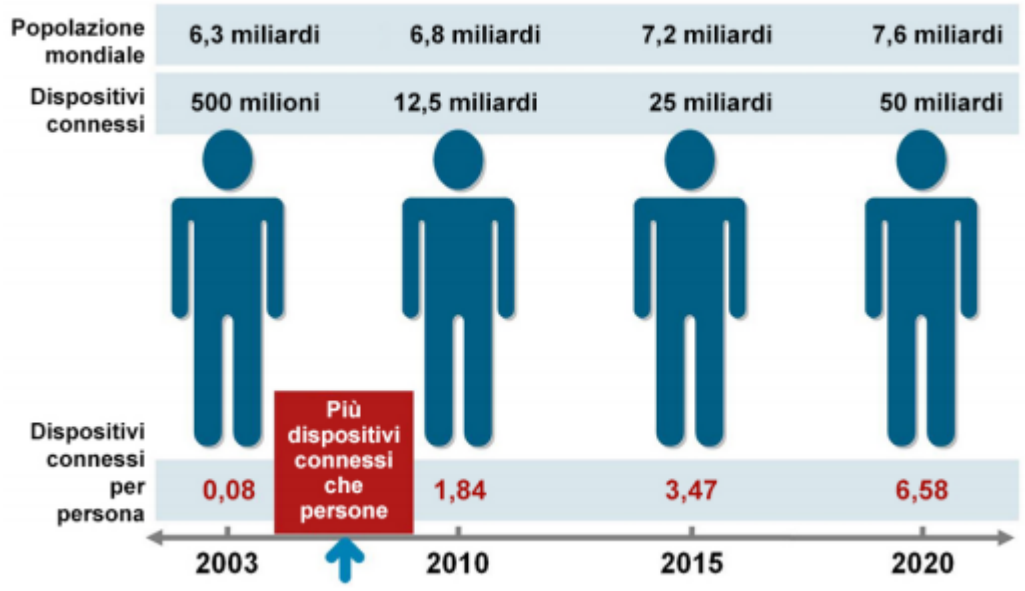

Figura 1.2: Crescita dei dispositivi connessi

1.2 Scenari applicativi

Il fatto che gli oggetti possano comunicare tra loro, dà potenzialmente il via a tantissimi scenari applicativi diversi. Molteplici sono i settori in cui si vuole intervenire con miglioramenti. Gli esseri umani vogliono vivere in salute, vogliono comfort, vogliono salvaguardare le risorse del pianeta e tanto altro. Grazie alla capacità di IoT di rilevare, acquisire, trasmettere e analizzare dati, questo diventa sempre più possibile. IoT quindi interviene dove si ha bisogno di migliorare la qualità della vita.

1.2.1 Logistica

Le imprese di logistica hanno bisogno si tenere monitorate le spedizioni, per sapere dove sono i beni in ogni momento e per ottimizzare i processi di spedizione. Sfruttando IoT, è possibile facilitare tutto questo. La tecnologia RFID viene largamente usata in questo ambito, e verrà trattata nel capitolo successivo. In breve, questa tecnologia permette a dei microchip(presenti in tag RFID) di inviare informazioni a un lettore tramite comunicazione wireless. Grazie a questa tecnologia le persone possono identificare, tracciare e monitorare gli oggetti che possiedono tag RFID. Si può pensare quindi di mettere tag RFID negli oggetti da monitorare. Facendone uno scan, i dati racconti possono essere inviati, sfruttando ad esempio reti wireless, a dei dispositivi fissi o mobili che raccolgono i dati. Questo torna utile sia alle imprese di logistica sia alle persone che vogliono monitorare un ordine. In questo scenario però emergono dei possibili problemi. Si considerino grandi volumi di dati. Può succedere che vengano raccolti dati incompleti (o che manchino dei dati) a causa della distanza dei collettori dei dati dalle sorgenti. Inoltre può anche capitare che ci siano collisioni durante la trasmissione dei dati. Inoltre bisogna far fronte al fatto che gli utenti (che per esempio vogliono controllare la posizione di un ordine) useranno sicuramente dispositivi di tipo diverso. A tal scopo usare middleware è fondamentale, ovvero uno "strato" tra il sistema operativo e le applicazioni, volto a permettere l'interoperabilità tra dispositivi di tipo diverso, e a fornire servizi alle applicazioni, migliorando la qualità del servizio.

1.2.2 Salute

IoT può essere di grande aiuto in campo medico. Ad esempio si può pensare di monitorare i pazienti con dei dispositivi wearable, che rilevano costantemente i loro segnali vitali o i loro spostamenti. Si può fare anche uso di dispositivi come telecamere e microfoni, per monitorare i suoni, le posture e i movimenti dei pazienti. Inoltre è utile usare dispositivi che possano eseguire dei controlli ad esempio sulle dosi di medicinali da somministrare, identificare bambini appena nati per evitare disguidi e altro. Generalizzando, dispositivi che eseguano dei controlli, in modo da sopperire a eventuali errori umani. Si consideri anche il caso di emergenze: monitorare da remoto lo stato di salute può facilitare i primi soccorsi, in modo tale che si possa comunque intervenire anche se nessun essere umano è fisicamente presente al momento dell'emergenza. Inoltre ci possono essere anche vantaggi per gli anziani, che sono sempre di più a causa dell'invecchiamento generale della popolazione mondiale. Grazie a dispositivi indossabili, si possono rilevare i parametri vitali di un individuo, inviare un avviso a un professionista dell'assistenza sanitaria al raggiungimento di un determinato valore o di rilevare quando una persona è caduta e non riesce a rialzarsi.

1.2.3 Città intelligenti

IoT può abbracciare anche città intere (le cosidette città intelligenti). Anche se non c'è una definizione di città intelligente accettata e condivisa da tutti, il suo scopo è di fare un miglior uso delle risorse pubbliche, aumentando la qualità dei servizi offerti ai cittadini, e nel contempo ridurre i costi della pubblica amministrazione. Vari aspetti di una città possono trarre vantaggi da IoT, come i trasporti, parcheggi, manutenzione delle aree pubbliche, scuole e altre strutture pubbliche, raccolta dei rifiuti e tanto altro. Si cerca quindi di costruire un IoT urbano, ovvero un'infrastruttura di comunicazione che fornisce un accesso unificato e semplice a svariati servizi. Di seguito verranno descritti brevemente i principali aspetti di una città intelligente:

- *Trasporti*: IoT risulta molto utile nel campo dei trasporti. Equipaggiando automobili, autobus e altri mezzi di trasporto con sensori, attuatori, è possibile realizzare sistemi di controllo per le collisioni e non solo. Se si vuole monitorare il traffico, si possono sfruttare i sensori e il GPS installato nelle auto più moderne, in combinazione a

sensori che monitorano l'inquinamento atmosferico e acustico in una determinata rotta. Grazie a queste informazioni, le persone possono sapere (grazie a dispositivi mobili) quale percorso conviene fare e tanto altro. E' possibile anche raccogliere dati sui parcheggi disponibili, piazzando sensori, in modo tale da informare i cittadini dei parcheggi con posti disponbili e di quelli invece pieni, con tutti i vantaggi che ne conseguono, ovvero meno traffico (quindi meno inquinamento) e meno stress per chi cerca parcheggio.

- *Ambiente*: Nel settore ambientale, si può fare largo uso di sensori (umidità, temperatura, pressione..) per raccogliere dati, che possono anche essere usati per prevedere eventualmente terremoti, incendi o controllare aree a rischio (discariche, impianti indistriali e chimici). Sensori possono essere usati anche per monitorare i livelli di inquinamento (sia atmosferico ma anche acustico). Questo permette alle autorità competenti di intervenire in modo da ridurre l'inquinamento, ma permette anche agli sportivi di sapere, tramite smartphone e relative applicazioni, di conoscere i luoghi più salutari dove allenarsi. Questi sensori possono essere usati anche nei beni di nutrimento come carne e frutta, per monitorare il loro stato durante il trasporto ai supermercati e assicurarne la qualità. Un esempio può essere ricercato anche nel settore dell'allevamento, in quanto si sta cominciando a usare IoT per monitorare gli allevamenti di bestiame. Sparked, una startup olandese, impianta sensori nelle orecchie dei bovini per consentire agli allevatori di tenere sotto controllo le loro condizioni di salute e di monitorarne gli spostamenti.

- *Ambienti smart*: IoT può migliorare sensibilmente la comodità della vita, a casa e in ufficio. Si possono distribuire sensori e attuatori, in modo da realizzare sistemi per il controllo della temperatura (i sensori rilevano la temperatura corrente, e gli attuatori cercano di mantenerla intorno al valore desiderato), controllo della luminosità di una stanza in base alla luce proveniente dall'esterno, controllo di emergenze (ad esempio sensori per rilevare fumo), controlli energetici (ad esempio ridurre i consumi dell'energia elettrica spegnendo una lampadina se non necessaria) e tanto altro. Tutto questo contribuisce a migliorare la comodità della vita, in quanto riduce le azioni che una persona deve

fare, allo stesso tempo rendendo la vita più sicura e riducendo anche le spese.

- *Industria*: I vantaggi si hanno anche in campo industriale, nel campo dell'automazione dei processi produttivi. Per esempio si può dividere le fasi di produzione di un oggetto in più stadi. Quando uno stadio viene raggiunto, viene letto un tag RFID da un lettore RDIF. Viene generato un evento, e un robot viene notificato. In questo modo il robot sa, in base all'evento, che azioni deve compiere in base allo stadio di produzione relativo.

- *Manutenzione edifici*: IoT può intervenire anche nella manutenzione di edifici (per esempio storici storici), monitorando continuamente le condizioni di essi e identificando le aree più soggette a agenti esterni. L'IoT urbano fornisce un database distribuito che contiene i dati raccolti dai sensori. Tutte queste misure contribuiscono a ridurre la necessità di controlli da parte dell'uomo, riducendo così le spese (ovviamente non considerando gli inevitabili costi iniziali per l'infrastruttura).

- *Raccolta rifiuti*: Sensori possono essere anche piazzati nei bidoni della spazzatura, in modo che i netturbini possano ottimizzare la tabella di marcia in base a quanto sono pieni.

- *Consumo energetico*: IoT può anche fornire un modo per monitorare il consumo energetico di tutta la città, rendendone disponibili i dati alle autorità e ai cittadini, in modo da tenerli sotto controllo, magari conoscendo le zone che consumano di più.

- *Illuminazione*: se i lampioni lungo le strade sono in grado di percepire la luce (con dei sensori), in base alla relativa quantità o all'ora della giornata si può regolare l'illuminazione, riducendo così i consumi e le spese.

- *Ticket mobili*: poster e pannelli possono essere equipaggiati con tag NFC (Near Field Communcation, tecnologia trattata nel capitolo successivo), codici visuali etc. Con uno scan da parte dello smartphone, viene aperto il browser che visualizza un sito web con delle informazioni a riguardo. Si pensi ad esempio alle informazioni su un'opera

d'arte in un museo, oppure alle informazioni relative a un treno (posti disponibili, prezzo biglietti). Tag possono anche essere usati nelle mappe turistiche, per reperire informazioni su hotel, luoghi da visitare e così via.

Concludendo, sostenuto l'investimento iniziale per creare l'infrastruttura per l'IoT urbano, i successivi vantaggi saranno molteplici in diversi campi. Miglior comfort generale e riduzione sensibile degli sprechi, con conseguenti ingenti vantaggi economici.

1.2.4 Altro

IoT può essere utilizzato anche in altri campi:

- *Furti e smarrimenti*: un oggetto, grazie alla rete Wireless, al GPS, o a un tag RFID, memorizza ciclicamente la propria posizione. Così in caso di furti o smarrimenti, una persona può conoscere la posizione da remoto. Sistemi più elaborati permettono di notificare automaticamente un utente se l'oggetto si allontana troppo da un luogo prestabilito.

- *Social network*: dei dispositivi generano eventi in base alla nostra posizione e altre informazioni, in modo da condividerle automaticamente sui social network.

- *Scenari futuristici*: sensori e altri dispositivi vengono impiantati nell'organismo umano, potendo così non solo misurare i parametri vitali, ma anche somministrare farmaci quando necessario in modo autonomo.

1.3 Problematiche

Gli scenari applicativi e le possibilità che IoT offre sono sicuramente innumerevoli e interessanti, ma ovviamente ci sono delle problematiche da tenere in considerazione. Sono tanti i vincoli e i requisiti tecnologici a cui bisogna far fronte per ottenere i risultati preposti. Le ricerche nel campo della microelettronica hanno reso possibile lo sviluppo di dispositivi sempre più piccoli, con tutti i vantaggi che ne conseguono. Ma l'effetto collaterale

di questo sviluppo è proprio che a causa delle ristrette dimensioni ci saranno limitate risorse computazionali, energetiche e di memoria. L'utilizzo di IoT inoltre comporta delle problematiche di sicurezza e di privacy. Bisogna controllare che le informazioni personali vengano tutelate a dovere, affinchè sia garantito appunto il diritto alla privacy. Un altro aspetto critico è che più IoT si fa strada nella nostra vita, più noi dipendiamo dalla tecnologia. Se tutto ruota attorno agli oggetti e alla connessione internet, in caso di blackout, sabotaggi ma anche errori di progetto, ci sarebbero gravissimi danni all'economia. In questa sezione verranno quindi analizzate le problematiche di tipo sociale (in particolare per quanto riguarda la tutela delle informazioni e la sicurezza nei confronti di attacchi) e le problematiche di tipo tecnologico insieme ai vincoli da rispettare.

1.3.1 Sicurezza e privacy

Il fatto che noi viviamo come se fossimo nodi della rete, unito al fatto che dati relativi alla nostra vita quotidiana vengono continuamente raccolti (e qualche volta anche esposti al pubblico) possono creare forti e seri problemi di sicurezza e privacy. La presenza di tag negli oggetti può essere nascosta agli utenti, i cui dati, senza rendersene conto, vengono raccolti. Per evitare che vengano raccolti dati senza consenso, o che vengano usati in modo sbagliato, servono dei requisiti:

- *Resistenza agli attacchi*: il sistema deve essere in grado di resistere agli attacchi e anche di aggiustare da solo i fallimenti dei nodi.
- *Autenticazione dei dati*: le informazioni recuperate devono essere autenticate.
- *Controllo dell'accesso*: chi fornisce i dati deve implementare il controllo d'accesso sui dati forniti.
- *Privacy dell'utente*: per proteggere i dati, devono essere prese misure che solo chi li fornisce è in grado di capire, in modo che il tutto sia il meno comprensibile possibile dall'esterno.

Per cercare di soddisfare questi requisiti, sono state sviluppate alcune tecnologie, chiamate Privacy Enhancing Technologies (PET), che vengono di seguito brevemente introdotte:

- *Virtual Private Networks (VPN)*: reti instaurate tra un numero ristretto di persone. In questo modo quel numero ristretto di persone può garantire la privacy e l'integrità dei dati raccolti.

- *Transport Layer Security (TLS)*: insieme di protocolli, operanti sopra il livello di trasporto, che garantiscono l'integrità dei dati.

- *DNS Security Extension (DNSSEC)*: estensioni dei DNS che garantiscono autenticazione e integrità dei dati forniti dai sistemi DNS, facendo uso della crittografia. Non garantiscono però la riservatezza e la disponibilità dei dati.

- *Onion Routing*: il traffico Internet viene criptato da più sorgenti e su più livelli, sfruttando gli onion ruoters (con le proprie chiavi) presenti lungo il percorso. Questo ovviamente diminuisce però la latenza e le performance.

- *Private Information Retrieval (PIR)*: protocolli che nascondono l'utente e l'informazione che esso richiede ad esempio da un database.

Anche le reti peer to peer possono aiutare nella sicurezza dei dati, oltre ad essere preferibili per quanto riguarda scalabilità e performance delle applicazioni. Per concludere, vengono ora illustrati in breve alcuni tipi di problematiche e attacchi alla sicurezza:

- *Clonazione di oggetti*: non è un vero e proprio problema di sicurezza, ma può causare danni economici ai produttori degli oggetti. Durante la fase di costruzione di un certo oggetto, può succedere che un malintenzionato riesca a clonarlo e magari venderlo a un prezzo inferiore dell'originale.

- *Sostituzione malevola di oggetti*: quando un sistema di oggetti viene installato, può succedere che alcuni di essi non siano genuini, ma siano di una peggiore qualità. Tuttavia verrebbero comunque riconosciuti e funzionerebbero col sistema. Questo ridurrebbe sicuramente i costi, ma aumenterebbe possibilità di fallimenti e diminuirebbe la sicurezza dei dati.

- *Attacco Eavesdropping*: quando gli oggetti comunicano attraverso la rete, possono essere suscettibili a intercettazioni. Se il canale di comunicazione non è sufficientemente sicuro, può succedere che chi compie l'attacco riesca a ottenere le chiavi di cifratura che servono per decodificare i dati che vengono trasmessi.

- *Attacco Man in the middle*: la fase di comunicazione può essere anche vittima dell'attacco man in the middle. Questo attacco può essere effettuato quando la sicurezza del protocollo di impostazione delle chiavi dipende dall'assunzione che nessun terzo sia in grado di piazzarsi tra le due entità che stanno eseguendo il protocollo. Un terzo quindi potrebbe porsi tra due entità che comunicano (senza che lo vengano a sapere) e ottenere i dati scambiati, o addirittura interagire con le due entità.

- *Attacco di rimpiazzamento del firmware*: quando un oggetto è in fase di manutenzione, può succedere che il suo software venga aggiornato per aggiungere nuove funzionalità. Un terzo potrebbe infiltrarsi e sostituire l'aggiornamento con un software malevolo modificato.

- *Attacco all'instradamento*: l'instradamento delle informazioni in rete può essere alterato, rimpiazzato e tanto altro. Alcune tipologie di questo tipo di attacco sono le seguenti (non viene descritto per semplicità e sinteticità il modo in cui si possano realizzare):

  - *Attacco Sinkhole*: chi attacca dichiara di possedere un percorso migliore verso la stazione base (di una rete di sensori - argomento trattato nel capitolo successivo), permettendogli così di fare ciò che vuole coi dati che passano in quel percorso.
  - *Instradamento selettivo*: chi attacca instrada pacchetti di dati selettivamente e ne scarta altri.
  - *Attacco Wormhole*: chi attacca riceve dei pacchetti di dati in un punto della rete, li indirizza a un altro punto della rete, in modo da modificare il comportamento e le statistiche della rete.
  - *Attacco di Sybil*: chi attacca presenta identità multiple agli oggetti della rete.

- *Attacco della negazione del servizio*: gli oggetti hanno poca memoria e risorse limitate. Sono quindi vulnerabili all'esaurimento delle risorse. Chi attacca può mandare continue richieste agli oggetti in modo da fare esaurire le loro risorse.

1.3.2 Requisiti e vincoli tecnologici

Gli scenari applicativi di IoT sono sicuramente interessanti, tuttavia le problematiche relative alla tecnologia non sono trascurabili. Il primo evidente vincolo che si può individuare è il fatto che i dispositivi in gioco abbiamo risorse computazionali e memoria limitate. A parte questo problema, unito al fatto del contenimento dei costi del materiale, se ne possono individuare altri:

- *Indirizzabilità*: a causa dell'elevato numero di oggetti che entrano in gioco in IoT, la capacità di indirizzamento di IPv4 non è più sufficiente. IPv4 usa 32 bit per gli indirizzi, quindi ce ne possono essere massimo 2^{32} diversi. Gli ultimi indirizzi IPv4 sono stati assegnati nel Febbraio 2010. Il passaggio a IPv6 (nel frattempo si stanno adottando situazioni temporanee) risolverà il problema, in quanto si passerà da 32 bit a 128 bit per gli indirizzi. IPv6 inoltre porterà miglioramenti in termini di sicurezza e semplificazioni nella gestione delle reti grazie a funzionalità di autoconfigurazione.

- *Scalabilità*: gli oggetti coinvolti in IoT sono tanti, anche se comunicano principalmente con quelli del loro ambiente locale. La comunicazione deve essere garantita sia in piccola scala sia in larga scala.

- *"Arrive and operate"*: dispositivi mobili eventualmente aggiunti dopo la formazione iniziale del sistema, non devono aver bisogno di configurazione, ma devono essere in grado di stabilire connessioni autonomamente con gli altri oggetti già presenti.

- *Interoperabilità*: gli oggetti sono di natura diversa (per esempio possono avere requisiti di larghezza di banda diversi, o hardware diverso). Questo implica la necessità di standard, in modo che oggetti di tipo diverso possano comunicare tra loro.

- *Volume di dati*: in particolare nelle reti di sensori, sarà presente una grande quantità di dati, che in diversi casi va memorizzata .
- *Interpretazione dei dati*: i dati raccolti dai sensori vanno interpretati in maniera accurata. Dai dati grezzi prodotti dai sensori, bisogna elaborare informazioni sensate.
- *Scoperta dei servizi*: l'ambiente di IoT è dinamico. E' quindi necessario che i servizi adatti per determinati scopi siano automaticamente identificati.
- *Complessità del software*: anche se il software negli oggetti dovrà funzionare con risorse minime, come in un convenzionale sistema embedded, è necessaria una complessa struttura software per gestirli e fornire loro servizi affinchè svolgano correttamente il loro compito.
- *Tolleranza ai guasti*: i cambiamenti di contesto in IoT sono molto frequenti. E' quindi necessario che gli oggetti si possano automaticamente adattare a questi cambiamenti. Inoltre è necessario fare forte uso della ridondanza, in modo da sopperire a guasti (che possono essere molto frequenti) e garantire il funzionamento del sistema in ogni caso.
- *Fonti di energia*: probabilmente una delle maggiore problematiche. Gli oggetti nella maggior parte dei casi sono mobili, quindi non hanno sempre la possibilità di essere collegati a una fonte di energia. Le batterie inoltre sono pesanti e grandi. Servono sistemi per produrre energia dall'ambiente circostante, in modo da rendere i dispositivi autosufficienti (ad esempio sfruttando la luce, il vento), che sfortunatamente, seppure ci siano progressi, non sono ancora in grado di risolvere il problema. E' necessario quindi anche ridurre la quantità di energia richiesta dagli oggetti.

Da tutte queste problematiche ne emerge il fatto che usare un middleware, ovvero un software che risiede tra il sistema operativo e le applicazioni, è necessario in sistemi di questi tipi, in modo tale da garantire interoperabilità e migliore gestione delle risorse.

Capitolo 2

Tecnologie abilitanti

Lo scopo di questo capitolo è analizzare le tecnologie più usate in IoT. Prima di procedere, è opportuno fare una considerazione. Il successo o meno di questa visione dipende dagli standard e dai middleware per diversi motivi: garantire una alta qualità del servizio agli utenti, e in particolare per garantire interoperabiità tra dispositivi di tipo diverso, compatibilità, affidabilità e tanto altro. I middleware verranno trattati più approfonditamente nel capitolo successivo. Con degli standard precisi, gli sviluppatori possono implementare applicazioni e servizi che possono essere distribuiti in larga scala, risparmiando tempo e costi sulla manutenzione a lungo termine. Anche la velocità con cui IoT migliorerà e crescerà sarà influenzata da tutto questo. Verranno introdotte alcune tecnologie per la comunicazione wireless come Wi-Fi, Bluetooth e iBeacon. Successivamente ci si focalizzerà, approfondendo di più, sulle reti di sensori wireless (parlando anche di Zigbee) e sulla tecnologia RFID. Di questi ultimi verranno analizzati nel capitolo successivo anche gli aspetti relativi ai middleware. Infine si parlerà di NFC.

2.1 Alcune tecnologie per la comunicazione wireless

Di seguito verranno introdotte alcune delle tipologie di comunicazione wireless più usate, ovvero Wi-Fi, Bluetooth e iBeacon. Ovviamente non sono le uniche presenti, e nei capitoli successivi ne verranno introdotte altre, come

i sensori wireless (e ZigBee per la comunicazione a corto raggio), RFID e NFC.

2.1.1 Wi-Fi

Wireless Fidelity (Wi-Fi) è una tecnologia per la comunicazione wireless che utilizza le onde radio basata sugli standard IEEE 802.11a/b/g per le wireless local area network (WLAN). Permette di avere l'accesso a internet sfruttando dei punti di accesso. I client non comunicano tra di loro direttamente, ma sfruttano degli Access Point connessi a rete cablata che si occupano di gestire la trasmissione dei dati. Un BSS (Basic Service Set) è formato dal punto di accesso e dai client connessi. I BSS possono comunicare tra di loro grazie agli Access Point. Un insieme di più BSS si chiama ESS (Extended Service Set) e funziona grazie a un componente chiamato sistema di distribuzione, grazie al quale IEEE 802.11 può creare una rete ESS di arbitraria dimensione e complessità. Wi-Fi supporta la mobilità, e i client possono muoversi liberamente attraverso la propria rete. La rete però può funzionare anche in una modalità ad hoc che permette ai client di comunicare tra loro senza passare per gli Access Point. L'insieme dei client comunicanti si chiama in questo caso IBSS (Independent Basic Service Set).

2.1.2 Bluetooth e Bluetooth Low Energy

Bluetooth, anche conosciuto come standard IEEE 802.15.1, è basato sulle onde radio e riguarda la comunicazione a corto raggio (massimo qualche decina di metri). E' un sistema molto economico. Viene ampiamente usato nei cellulari, ma anche nei computer, dove serve per eliminare la necessità di utilizzare cavi per connettere al computer dispositivi come stampanti, joystick e altro. Vengono formate delle wireless personal area network (WPAN). Ci sono due tipologie di connettività: la piconet e la scatternet. Piconet è una WPAN formata da un dispositivo Bluetooth che agisce da master e altri dispositivi Bluetooth che agiscono come slave. Viene quindi creato un canale di comunicazione e gli slave vengono sincronizzati in base al clock del master, che ovviamente gestisce la comunicazione (che può essere punto-punto o punto-multipunto). Scatternet è un insieme di piconet che si sovrappongono vicendevolmente sia spazialmente sia temporalmente. In questo modo un dispositivo può partecipare a più piconet nello stesso momento. Per ri-

durre drasticamente i consumi di Bluetooth, è stato sviluppato Bluetooth Low Eneegy (BLE). In questo modo si può equipaggiare piccoli dispositivi con BLE, rendendolo particolarmente adatto a IoT. BLE si distingue dal Bluetooth tradizionale anche per il suo costo più ridotto.

2.1.3 IBeacon

IBeacon è uno standard definito da Apple che permette alle applicazioni iOS e Android di ottenere dati dai dispositivi iBeacon ricevendo da loro messaggi. E' rivolto principalmente a problemi di localizzazione. Questi dispositivi emettono continuamente messaggi in un formato ben preciso, che possono essere captati da smartphone o altri dispositivi con BLE. I messaggi sono composti da:

- *Universally Unique Identifier-UUID*: stringa di 16 bit usata per differenziare tra di loro i gruppi di beacon diversi. Ad esempio se un'azienda ha una rete di beacon, questi avranno lo stesso UUID.
- *Major*: stringa di due bit usata per distinguere sottogruppi all'interno del gruppo più grande.
- *Minor*: stringa di due bit che serve per identificare univocamente il singolo beacon.

Un possibile scenario applicativo può essere il seguente: un utente si trova all'interno di un negozio, e un'applicazione dedicata sul suo smartphone si mette a captare beacon. A seconda dei dati captati l'utente viene poi notificato di informazioni rilevanti che possono tornargli utili.

2.2 WSN - Wireless Sensor Network

I sensori sono dei dispositivi di dimensioni ridotte che permettono di raccogliere dati relativi all'ambiente circostante. Una rete di sensori wireless (WSN), è un sistema complesso dovuto alla cooperazione tra diversi nodi sensore. E' una rete ad hoc, poichè viene sviluppata appositamente per un determinato problema, e ha dei vincoli energetici, dato che i nodi sensore hanno risorse ed energia limitati. E' un tipo di sistema embedded, in quanto

costituito da dispositivi elettronici in grado di svolgere autonomanente determinate operazioni, interagendo con l'ambiente e in grado di cooperare tra loro grazie a oppurtune interfacce di comunicazione. Per analizzare questo argomento verrà usato un approccio top-down, quindi verranno mostrate per prima cosa le architetture più usate di una rete di sensori wireless, e in seguito l'architettura di un singolo nodo sensore.

2.2.1 Architettura di una WSN

Il principale modo di organizzare una rete di sensori è di avere nodi sensori multipli e un'unica stazione base. Si usa poi un server per comunicare le informazioni fuori dalla rete, come mostrato in figura 2.1. Ci sono diversi modi per organizzare una WSN, tra i quali il modello a grappolo, a maglia, a stella e ad albero. Verrà trattato il modello a grappolo poichè presenta diversi vantaggi, tra i quali l'essere più adatto a reti di grandi dimensioni. Tutta la rete può essere divisa in piccole parti dette grappoli. Ogni grappolo possiede un certo numero di nodi sensori e una testa. Il nodo sensore comunica con la testa, e la testa comunica con la stazione base. Il modello della rete a grappolo possiede due sottomodelli, chiamati Single hop model and Multi hop model.

- *Single-Hop Model*: i nodi sensori comunicano direttamente con la testa, e la testa comunica direttamente con la stazione base, come mostrato in figura 2.2.

- *Multi-Hop Model*: si fa uso di teste intermedie, divise in livelli in base alla distanza dalla stazione base: i nodi distanti dalla stazione base, comunicano con le teste del relativo grappolo, che a loro volta comunicano con teste intermedie più vicine che comunicheranno con la stazione base, come mostrato in figura 2.3.

Il modello single hop avrebbe teoricamente meno tempi di latenza rispetto al modello multihop, in quanto la comunicazione è diretta (ha meno passaggi intermedi). Tuttavia i sensori hanno range limitato, e spesso non è possibile applicare sempre il modello single hop. Organizzare la rete a grappolo migliora i tempi di latenza. Se usassimo un modello multi hop, ma senza l'organizzazione a grappoli (quindi con i dati che passano da nodo sensore a nodo sensore fino ad arrivare alla stazione base) i tempi sarebbero molto più elevati. Ci sono comunque degli effetti collaterali (anche se i

vantaggi che ne conseguono sono maggiori), in quanto più livelli intermedi ci sono, più aumentano i consumi, che sono proporzionali al quadrato della distanza.

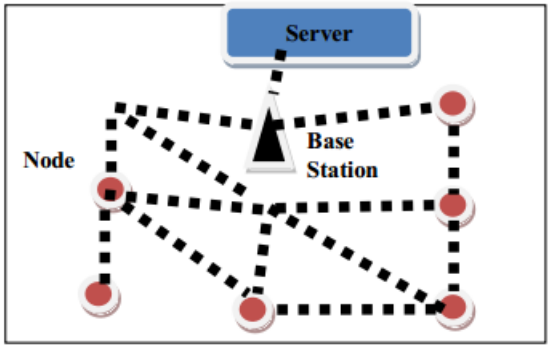

Figura 2.1: Architettura di base di una WSN

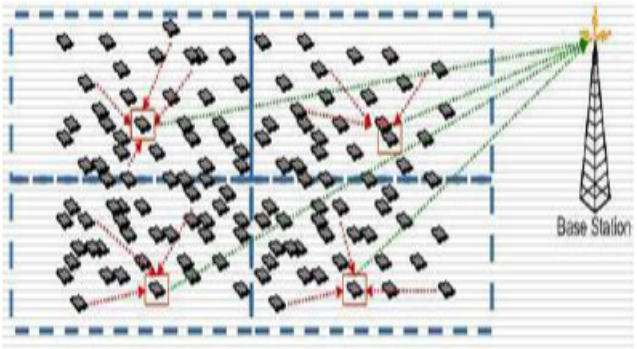

Figura 2.2: Single hop model in una WSN a grappolo

2.2.2 Architettura di un nodo sensore

La figura 2.4 mostra schematicamente l'architettura di un singolo nodo sensore. Di seguito ne verranno analizzati i blocchi.

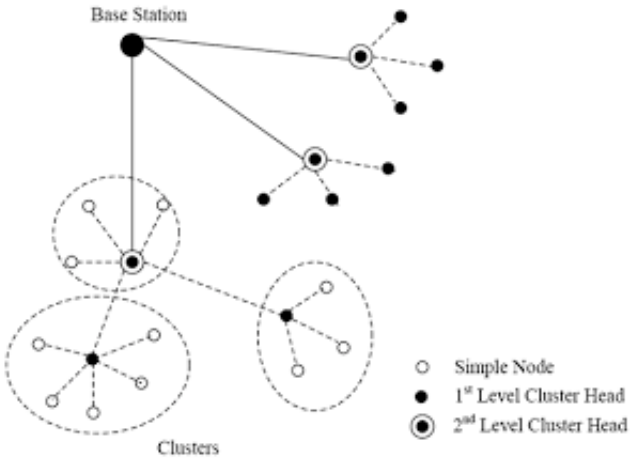

Figura 2.3: Multi hop model in una WSN a grappolo

- *Trasduttori e attuatori*: Un trasduttore è un dispositivo che può misurare grandezze fisiche e ambientali, trasformandole in un segnale elettrico a seconda del valore misurato. Solitamente (in modo non del tutto corretto) si pensa che siano i sensori a svolgere questa funzione, in realtà si dovrebbe chiamare sensore il nodo sensore complessivo. Un trasduttore può misurare temperatura, umidità, luce, pressione, sostanze varie e tanto altro. Non è però possibile agire sull'ambiente con questo tipo di dispositivi, è necessario usare attuatori (ad esempio bracci meccanici, valvole, sistemi d'allarme etc). A seconda del sistema che si vuole realizzare, si avranno diverse configurazioni di trasduttori e attuatori.

- *Unità di controllo e di elaborazione*: Qui è presente un microprocessore o microcontrollore (CPU). Si può realizzare usando o microcontrollori a basso consumo, ma anche FPGA (Field Programmable Gate Array), DSP (Digital Signal Processor) o altri circuiti logici. E' presente anche un convertitore analogico digitale, poichè è necessario convertire i segnali elettrici dei trasduttori in forma digitale, affinchè possano essere elaborati. Per dare comandi agli attuatori è invece necessario usare un convertitore digitale analogico (che trasforma il segnale digitale in segnale elettrico).

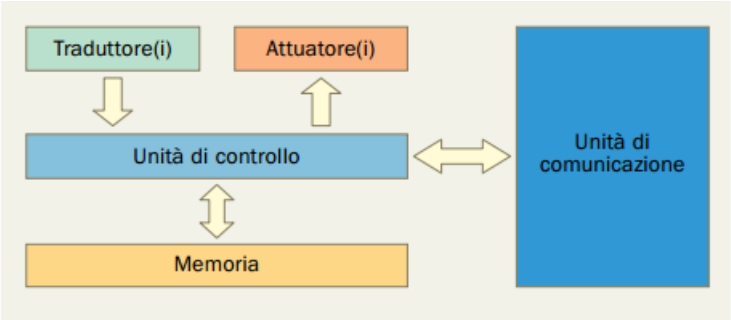

Figura 2.4: Architettura di un nodo sensore

- *Memoria*: Nel caso sia presente un microprocessore nell'unità di controllo, vengono usati blocchi di memoria ROM e RAM (se usassimo un microcontrollore non sarebbe necessario, in quanto già integrati) per salvare i dati raccolti e contenere le istruzioni di codice. Tuttavia le memorie usate sono di piccola dimensione (poche decine di kbyte), in quanto incidono molto sul consumo energetico.

- *Unità di comunicazione*: I nodi sensori devono poter comunicare tra loro. Si usano tipicamente segnali radio. Il chip radio però è proprio il componente che consuma di più, quindi si sacrifica la capacità trasmissiva in modo da avere meno consumi.

- *Software*: A parte l'hardware, la piattaforma deve ospitare anche il software necessario per la gestione della comunicazione tra i nodi e lo svolgimento del compito del sistema da realizzare. Il sistema operativo da usare deve soddisfare dei requisiti:

  - ridotta occupazione di memoria
  - basso consumo energetico da parte dei processi
  - consumo praticamente nullo durante l'inattività
  - supporto al multithreading
  - supporto efficiente (in termini di consumo energetico) ai protocolli di rete

2.2.3 Zigbee

A causa del loro limitato raggio di comunicazione, le reti di sensori hanno difficoltà a comunicare col mondo esterno. Inoltre va considerata la grande eterogeneità dei dispositivi, che sono prodotti spesso da aziende diverse. Risultano quindi necessari standard e protocolli. Alla base delle reti di sensori c'è lo standard IEEE 802.15.4. Questo è uno standard che definisce le caratteristiche del livello fisico e del livello MAC (Medium Access Control) per le WPANs (Wireless Personal Area Networks). Si riferisce quindi a reti con basso tasso di dati, con bassa complessità e bassi consumi energetici. Prendendo IEEE 802.15.4 come base, la ZigBee Alliance ha definito i livelli superiori con lo standard ZigBee. Questo standard serve per la comunicazione di dispositivi a corto raggio. Le WPANs sono costituite da dispositivi, che possono essere categorizzati come di tipo fisico e di tipo logico. I dispositivi fisici possono a loro volta essere classificati in Full Function Device (FFD) e Reduced Function Device (RFD). Ogni dispositivo può comportarsi come un nodo sensore o un nodo di controllo. Le funzioni di instradamento sono eseguite solo dai FFDs. A seconda della loro posizione nella rete i FFDs possono possedere uno o più dispositivi figli e effettuano le funzioni di routing per loro. I RFDs non possono fare instradamento e non possono avere dispositivi figli. I dispositivi logici si suddividono in coordinatori, router e terminali. Il coordinatore è la radice della rete, e affinchè una rete possa essere costruita ce ne deve essere esattamente uno. Funziona anche da ponte per le altre reti. I terminali hanno funzionalità limitate e possono comunicare solo con i coordinatori e i router, e proprio per questo motivo possono dormire per molto tempo e quindi avere un ciclo di vita molto lungo. Lo stack di protocolli definito dalla ZigBee Alliance è mostrato in figura 2.5 Sono inclusi:

- *Livello fisico*: addetto alla ricezione, trasmissione modulazione di segnali.
- *Livello MAC*: deve accedere fisicamente alla rete, con meccanismi per la correzione di errore. Si occupa anche di trasmettere il beacon (si veda dopo l'elenco le tipologie di comunicazione beacon e non beacon).
- *Livello di rete e di sicurezza*: deve gestire i nuovi dispositivi e la crescita della rete. Inoltre svolge le funzioni di routing e possiede meccanismi per la sicurezza.

- *Application Support sub-layer (APS)*: fornisce i servizi necessari agli application objects e allo Zigbee device object necessari per farli interfacciare con il livello di rete (ad esempio le funzioni di richiesta, conferma e risposta). Ha il compito di effettuare la manutenzione di alcune tabelle, che contengono informazioni usate per stabilire la comunicazione tra dispositivi. Durante la fase di scoperta di nuovi dispositivi queste tabelle sono utilizzate anche per appunto identificarne di nuovi che possono operare.

- *Application Objects*: moduli di applicazioni che implementano una applicazione ZigBee.

- *Zigbee Device Object (ZBO)* : controlla e gestisce gli application objects. Ha il compito di determinare la natura dei dispositivi in una rete (per esempio FFD o RFD). Deve anche garantire rapporti sicuri tra dispositivi.

- *Application profiles*: il livello applicativo è definito dai profili applicativi, cioè linee guida per certi tipi di applicazioni conformi allo standard ZigBee.

La comunicazione in reti ZigBee può essere di due tipi:

- *Beacon*: un dispositivo aspetta segnalazioni (beacon) che sono periodicamente trasmesse dal coordinatore. Quando lo riceve passa da inattivo a attivo e controlla se il beacon è diretto a lui: se sì svolge le sue funzioni, sennò ritorna inattivo. Tra un beacon e l'altro i nodi possono andare in inattività per consumare meno energia.

- *Non beacon*: alcuni dispositivi restano accesi (aumentando così i consumi) e altri invece si risvegliano solo se stimolati.

Lo standard ZigBee però non supporterebbe le reti a grappolo descritte precedentemente, ma solo reti a stella, albero e a maglia. Il motivo è che se usassimo ZigBee con reti a grappolo, ci sarebbero molti problemi di collisione dei dati. La rete possiede il coordinatore, i router e i nodi terminali, come mostrato in figura 2.6. Il coordinatore e i router possono mandare le segnalazioni per sincronizzare gli altri nodi. I tipi di collisione possono essere due:

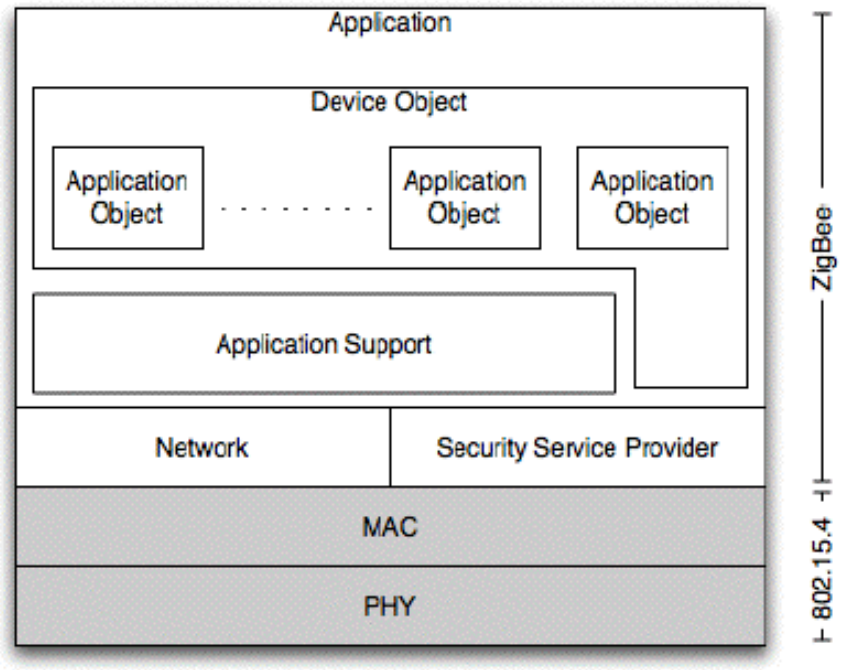

Figura 2.5: ZigBee e IEEE 802.15.4

- *Diretta*: avviene quando due o più coordinatori sono nel raggio di trasmissione l'uno dell'altro.

- *Indiretta*: avviene quando due o più coordinatori non si conoscono, ma hanno il range di trasmissione che si sovrappone.

Un esempio può essere questo: supponiamo che un nodo A debba sincronizzarsi con un coordinatore C1. Se A si trova anche nel raggio di un altro coordinatore C2 e magari riceve di poco prima la sua segnalazione, perde la sincronizzazione con C1. Esistono metodi per risolvere queste problematiche che sfruttano la divisione dei tempi e quindi ZigBee si può utilizzare anche con reti a grappolo, ma la loro trattazione esula dallo scopo di questa tesi.

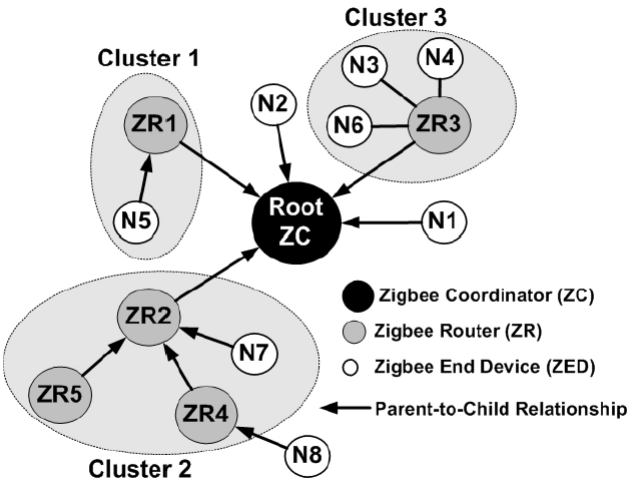

Figura 2.6: Modello a grappolo con ZigBee

2.3 RFID - Radio-Frequency Identification

RFID usa le onde radio, una forma di onda elettromagnetica. I primi sviluppi di questa tecnologia sono stati in ambito militare, ovvero nei radar (durante la seconda guerra mondiale). Veniva sfruttato il principio secondo cui le onde radio si riflettono sugli oggetti. Grazie a questo è possibile rilevare dimensione e posizione dell'oggetto. Ora verrà mostrato brevemente come è costituito un sistema RFID. Come mostrato in figura 2.7, ci sono tre componenti principali:

- *RFID tag*: E' un transponder a radiofrequenza di piccole dimensioni costituito da un circuito integrato (chip) con funzioni di semplice logica di controllo, dotato di memoria, connesso ad un'antenna ed inserito in un contenitore o altro. Il tag permette la trasmissione di dati a corto raggio senza contatto fisico. Salvo eccezioni, i dati contenuti nella memoria del tag sono limitati ad un codice univoco (identificativo). La struttura di un tag RFID è descritta più nel dettaglio nella sottosezione seguente.

- *Lettore RFID*: componente che interroga i tag RFID. Viene detto

anche interrogatore. E' un ricetrasmettitore (dotato di antenna) controllato da un microprocessore ed usato per interrogare i tag e ricevere le informazioni in risposta.

- *Sistema di gestione*: un sistema informativo che, quando esiste, è connesso in rete con i lettori. Tale sistema consente, a partire dai codici identificativi provenienti dai tag, di ricavare tutte le informazioni disponibili associate agli oggetti e di gestire tali informazioni per gli scopi dell'applicazione.

Il funzionamento di un generico tag RFID (dato che ci possono essere delle variazioni) è il seguente: quando il tag passa attraverso il campo elettromagnetico generato da un lettore, trasmette a quest'ultimo le proprie informazioni. Una volta che il tag ha decodificato come corretto il segnale dell'interrogatore, gli risponde riflettendo, mediante la sua antenna, e modulando il campo emesso dal lettore. I tag RFID possono essere fondamentalmente di quattro tipi.

- *Passivi*: sono quelli più largamente usati, in particolare a causa del loro basso costo e della loro versatilità. Contengono un microchip e un'antenna per ricevere/trasmettere segnali. Non sono alimentati internamente, infatti vengono ricaricati grazie ai segnali ricevuti dagli interrogatori (quando vengono appunto interrogati), perciò possono inviare segnali solo se interrogati. non possiedono un vero e proprio trasmettitore, ma reirradiano, modulandolo, il segnale trasmesso dal lettore e riflesso dalla propria antenna. Le distanze a cui possono operare sono, al massimo, dell'ordine di alcuni metri o di alcuni centimetri a seconda della frequenza operativa. Contengono un codice univoco per l'identificazione.

- *Attivi*: sono più potenti di quelli passivi, in quanto incorporano ricevitore e trasmettitore come i lettori. Sono alimentati internamente a differenza dai passivi, quindi possono anche contenere sensori. Possiedono memorie di dimensioni notevoli, spesso riscrivibili. Le distanze a cui possono operare dipendono da trasmettitore e batterie, e in genere sono, al massimo, dell'ordine di 200 metri. Hanno il vantaggio di poter realizzare sistemi che non riguardano solo la pura localizzazione, in quanto essendo autoalimentati, possono iniziare autonoma-

mente la trasmissione (a differenza dei passivi che possono solo essere interrogati).

- *Semipassivi*: si differenziano dai tag passivi in quanto i passivi devono usare parte dell'energia raccolta dai segnali degli interrogatori per l'alimentazione del microchip, mentre i semipassivi hanno il microchip alimentato internamente, in modo tale da usare tutta l'energia raccolta dai lettori per la comunicazione (che quindi può avvenire a una distanza più elevata, fino a qualche decina di metri).

- *Semiattivi*: si differenziano da quelli attivi in quanto possono eseguire trasmissioni solo quando vengono interrogati dagli interrogatori. Sono alimentati internamente, perciò una volta attivati dal lettore, possono continuare a funzionare autonomamente per un certo periodo di tempo.

E' da menzionare anche che i tag, inoltre, possono essere di tipo read-only o read-writable. Questi ultimi consentono, durante il loro uso, oltre alla lettura, anche la modifica o la riscrittura dell'informazione in essi memorizzata. Ovviamente quelli riscrivibili sono più costosi.

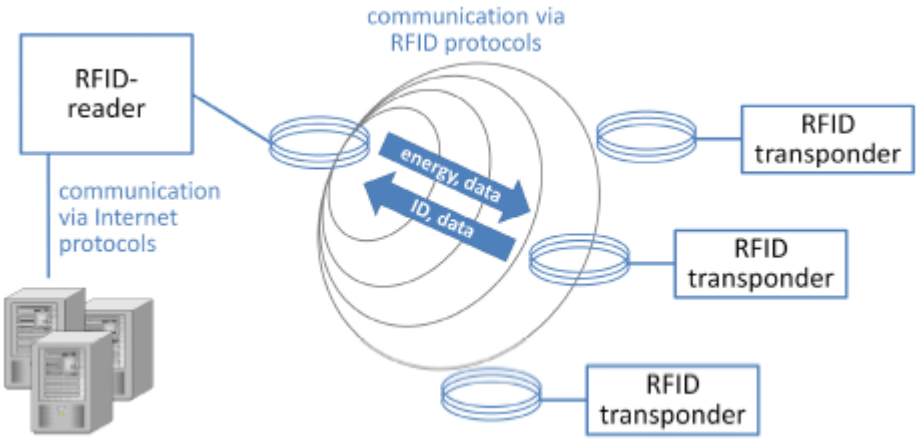

Figura 2.7: Tags RFID, lettore e sistema informativo

2.3.1 Architettura di un sistema RFID

Dopo aver analizzato i componenti di un sistema RFID, è necessario introdurre una possibile architettura di un generico sistema RFID, in modo da mostrare l'organizzazione dei componenti. Questo tipo di architettura permette di tracciare la posizione corrente dei tag RFID. Si assume che ogni lettore RFID possieda un indirizzo IP. Un nodo (ad esempio un computer) può determinare la posizione dei tag associando il numero RFID con l'indirizzo IP del lettore vicino al tag. L'architettura è mostrata in figura 2.8 I componenti presenti sono i seguenti:

- *Lettori RFID*: già introdotti. Permettono di recuperare informazioni dai tag.

- *Database delle posizioni*: base di dati che ha il compito di memorizzare la posizione corrente dei tag e in particolare del mapping tra il numero RFID del tag e l'indirizzo IP del lettore.

- *Agente RFID (RA)*: è un componente che si occupa di raccogliere e filtrare i dati raccolti ed è responsabile della procedura di registrazione delle posizioni. E' una entità logica che può essere integrata all'interno dei lettori RFID se questi possiedono risorse sufficienti o può essere indipendente.

- *Server delle posizioni*: server che accetta richieste di registrazione e di tracciamento. Quando riceve una richiesta di registrazione, questo salva le informazioni che riceve nel database delle posizioni, che verrà usato per gestire le posizioni dei tag. Ogni tag possiede la sua posizione di base. Quando questo si muove, deve effettuare una registrazione di posizione, in modo da informare il server che la sua posizione è cambiata, in modo che nel database possa essere salvata la nuova posizione. Se invece riceve una richiesta di tracciamento, il server controlla la posizione del tag nel database e risponde con la posizione corrente.

- *Server dei nomi*: è una base di dati che mantiene i mapping tra i tag RFID e gli indirizzi dei server delle posizioni (dove quindi saranno reperibili le informazioni relative al tag), in modo simile a come operano i server DNS. Ad esempio, il server dei nomi prende un tag

RFID come input e ritorna l'indirizzo IP del server delle posizioni come output. I server dei nomi possono essere organizzati in gerarchie per una migliore gestione e migliori prestazioni. Se un server locale non è in grado di risolvere una richiesta, questa viene delegata a un server di livello superiore. Questa procedura è utile, poichè quando un lettore interroga un tag, è possibile sapere dove sono salvate le informazioni riguardo a quel tag, e quindi recuperarle (ad esempio per fornirle all'azienda che deve tracciare dei prodotti).

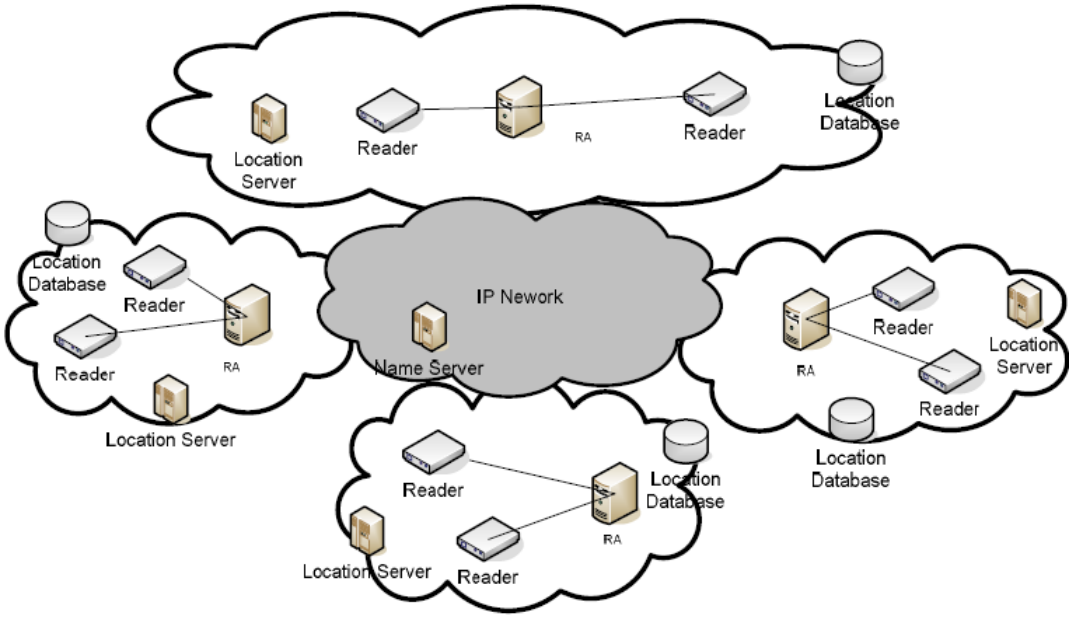

Figura 2.8: Architettura di un sistema RFID

2.3.2 Struttura di un tag RFID

Il tag RFID è un componente che, applicato a un oggetto o altro, permette la sua identificazione senza un collegamento diretto, grazie a segnali radio. Il tag è formato da tre componenti principali: il chip, l'antenna e l'inlay.

- *Chip*: è un circuito elettronico intelligente, che contiene la parte logica e la memoria del tag, e deve gestire gli scambi di informazioni tra il tag e il lettore.

- *Antenna*: apparato che riceve e trasmette i segnali radio da/verso i lettori. Va costruita in base a quanto i tag sono distanti dai lettori. Se il tag è passivo, l'antenna riceve il segnale dal lettore e lo utilizza per alimentare il chip, che modula il segnale ricevuto per fornire le informazioni di risposta. Se il tag è attivo l'antenna non usa il segnale per alimentare il chip, poichè è alimentato già dalla batteria.

- *Inlay*: struttura che collega il chip e l'antenna e li protegge. Ovviamente il progetto dell'inlay dipende dalle caratteristiche che si vuole conferire al tag (resistenza agli urti, a particolari agenti chimici, umidità etc). Si può realizzare semplicemente con plastica o anche con materiale cartaceo.

I tag possono anche contenere memoria (che si trova eventualmente nel chip, e indica quanti dati può memorizzare), e in base a quello che si vuole realizzare possono essere di tipologie diverse.

- *ROM (Read Only Memory)*: viene configurata dal produttore e non può essere modificata. Solitamente viene usata per memorizzare il codice di identificazione del tag. Risulta quindi poco costosa, dovendo memorizzare solo quello.

- *WORM (Write Once Read Memory)*: l'utente può riscrivere la memoria una volta sola, poi non è più possibile farlo. In questo modo il tag è personalizzabile e non serve l'intervento del produttore. Una volta programmata si comporterà come una memoria ROM.

- *EEPROM (Electrically Erasable Read Only Memory)*: va alimentata solo durante la scrittura e la lettura. Mantiene in memoria per anni le informazioni senza bisogno di essere alimentata, per questo è molto più costosa delle altre tipologie. L'utente può programmare solo una parte della memoria (quante volte lo desidera). Il resto non è modificabile e contiene il codice di identificazione del tag scelto dal produttore.

- *RAM (Random Access Memory)*: va alimentata costantemente per mantenere i dati in memoria. Può essere programmata un numero illimitato di volte. E' molto versatile perchè a un costo ridotto permette di memorizzare molti dati.

2.3.3 Standard

I tag e gli lettori devono interagire tra di loro facilmente e senza disturbare altri servizi a radiofrequenza. Bisogna quindi porre attenzione alle bande di frequenze entro le quali operare, alla larghezza di banda, alla potenza radio di emissione e controllare eventali emissioni di segnali spuri. Inoltre è fondamentale, a causa della grande eterogeneità dei dispositivi in commercio, che dispositivi prodotti da aziende diverse siano in grado di interoperare tra loro. A tal fine è necessario definire parametri vari, regole per la codifica e impacchettamento dei dati, la loro struttura, accordare la velocità di trasmissione e tanto altro. Sono quindi necessari degli standard. Per trasferire l'informazione attraverso lo spazio che separa l'interrogatore dal tag il segnale viene modulato su un segnale portante, di frequenza indicativamente compresa fra 100 kHz e 5.8 GHz. Si individuano quattro sotto intervalli.

- *Bassa frequenza (125-135 KHz)*: i tag sono accoppiati magneticamente con i lettori, il raggio del sistema RFID risulta quindi molto corto. L'interrogatore deve essere distante solo qualche centimetro dal tag. Sistemi a bassa frequenza quindi vengono usati maggiormente nel controllo degli accessi, nella sicurezza delle macchine, nell'identificazione di animali e altro.

- *Alta frequenza (10-15 MHz)*: Anche qui i tag sono accoppiati magneticamente con gli interrogatori. Questi tipi di sistemi possiedono queste caratteristiche:

  - raggio operativo al di sotto del metro (corto, anche se superiore al caso precedente)
  - scarsa sensibilità ai liquidi
  - sensibilità ai metalli
  - media velocità nelle operazioni di lettura/scrittura

Sistemi di questo tipo si usano solitamente nel controllo degli accessi e degli articoli. La banda utilizzata è 13.56 MHz (standard a livello mondiale).

- *Altissima frequenza (850-950 MHz)*: I tag sono accoppiati elettromagneticamente con gli interrogatori. Il raggio operativo è lungo, la velocità di lettura e scrittura è alta, la sensibilità ai liquidi e metalli è alta. I sistemi di questo tipo risentono particolarmente del fenomeno fisico della risonanza dell'acqua. Per questo motivo questi sistemi sono influenzati dalle condizioni ambientali (come umidità, e anche riflessioni prodotte da oggetti elettromagneticamente riflettenti). Per far fronte a questa problematica, i tag vanno inglobati con dei degli specifici packaging, per fare in modo che il sistema funzioni anche in ambiente ostile. I sistemi ad altissima frequenza sono quelli più utilizzati, e le frequenze di interesse sono intorno ai 900 MHz, anche se ogni paese possiede specifiche normative a riguardo.

- *Microonde (2.45-5,8 GHz)*: i tag sono accoppiati elettromagneticamente con gli interrogatori. Il range di funzionamento è molto ampio. Sistemi di questo tipo sono caratterizzati da una problematica chiamata "standing wave nulls", ovvero di zone morte all'interno del campo di lettura in cui non si ha accesso al tag. Il fenomeno è causato dalla ridotta lunghezza d'onda della radiazione a microonde (da 12 a 30 centimetri). Se il tag è immobile, potrebbe quindi trovarsi in una zona morta, impedendo il funzionamento del sistema. Sistemi di questo tipo quindi vanno impiegati in ambiti particolari. Un esempio è il sistema di pagamento del pedaggio in modo automatizzato (Telepass) in cui si opera alla frequenza di 5.8 GHz. I veicoli sono in movimento, risolvendo così il problema dello "standing wave nulls".

Generalizzando, un aumento della frequenza produce un incremento della velocità di lettura e scrittura, ma diminuisce la capacità di trasmissione in ambienti particolari in cui sono presenti ostacoli metallici (come nei magazzini in cui ci sono scaffali, contenitori e vari oggetti) o liquidi, anche se con il giusto packaging si può ridurre il problema. Bisogna quindi adottare il tipo di sistema adatto in base al relativo scenario applicativo. Questi appena elencati sono gli standard relativi alle frequenze. Per quanto riguarda gli standard per l'interoperabilità e la comunicazione tra lettori e tag prodotti

da aziende diverse, i primi sono stati introdotti da ISO/IEC. Successivamente anche EPCglobal ha introdotto i suoi standard. I tag RFID sono stati divisi in classi in base alla loro tipologia e alla frequenza di funzionamento. E' doveroso menzionare lo standard di livello mondiale Gen-2 di EPCglobal (che riguarda i tag passivi a altissima frequenza). E' basato su una architettura master-slave, in cui i lettori sono i master e i tag sono gli slave. Un lettore, per accedere a un tag, deve svolgere tre azioni:

- *Selezione*: il lettore seleziona dei gruppi (o singoli) nella popolazione dei tag in base a certi criteri (il tag non risponde al lettore).
- *Inventario*: il lettore identifica i tag selezionati. Il tag deve rispondere al lettore inviando il proprio codice identificativo (a differenza dell'operazione precedente in cui non rispondeva).
- *Accesso*: il lettore esegue operazioni di lettura/scrittura sul tag.
- *Altre operazioni*: svolgere le operazioni di interesse con il tag desiderato.

2.4 NFC - Near Field Communication

NFC è una tecnologia di comunicazione radio, proprio come la tecnologia Wi-Fi, Bluetooth e altre. Si contraddistingue da queste perchè opera solo a distanza di qualche centimetro (massimo 10). Più precisamente è una serie di specifiche e standard per l'accoppiamento induttivo a radio frequenza (13.56 MHz) per trasferire informazioni tra due dispositivi vicini. Non è una tecnologia nuova, ma sta cominciando a diventare importante in questi ultimi tempi, specialmente perchè si sta facendo largo nel mondo degli smartphone. Integrando chip e antenna NFC nello smartphone è infatti possibile realizzare sistemi di pagamento (avvicinando lo smartphone a degli appositi terminali nei negozi), sistemi di condivisione di dati tra dispositivi, accedere a contenuti digitali leggendo smart tags (piccoli chip di sola lettura che si possono trovare in poster e documenti, come badge o passaporti), sistemi di autenticazione (accedere a reti senza dover inserire password e tanto altro, magari avvicinando lo smartphone al router). NFC è una tecnologia nata dall'evoluzione della tecnologia RFID (quindi un dispositivo NFC può operare con sistemi RFID già esistenti), che permette di trasferire

piccole quantità di dati tra due dispositivi che si trovano a pochi centimetri di distanza l'uno dall'altro, senza la necessità di codici di accoppiamento. Il vantaggio su tutte le altre è quindi proprio la semplicità, perchè questa tecnologia permette di trasformare lo smartphone in una carta di credito senza fili e tanto altro. Il funzionamento è il seguente: il dispositivo che inizia la comunicazione usa un'antenna per emettere un segnale radio a 13.56 MHz, generando quindi un campo elettromagnetico. Se un dispositivo NFC è presente nel campo, viene attivato e quindi creato un canale di comunicazione. La comunicazione dovrà però avvenire solo con un destinatario, perciò in caso di presenza di più dispositivi si dovrà comunicare solo con uno alla volta. La comunicazione si dice attiva se sia chi inizia la comunicazione sia il destinatario generano il proprio campo, si dice invece passiva se solo chi inizia la comunicazione lo genera. La comunicazione è half-duplex, ovvero che un dispositivo riceve quando l'altro sta trasmettendo. Il bit rate può essere 106, 212 o 424 kbps, quindi non molto veloce, perciò vanno scambiate piccole quantità di dati. I dispositivi NFC si dicono attivi se generano loro il campo (quindi sono alimentati da batteria), e passivi se non lo generano e vengono alimentati dai campi generati da altri dispositivi. Riassumendo, un device NFC può funzionare in tre modalità diverse:

- *Modalità lettore/scrittore*: in questa modalità, il dispositivo NFC può leggere e modificare dati contenuti in tag NFC passivi (che si possono trovare ad esempio nei poster), permettendo all'utente di recuperare delle informazioni. Per quanto riguarda i consumi energetici, l'energia per generare il campo NFC deve essere fornita dal dispositivo attivo.

- *Emulazione di carta*: un dispositivo NFC può comportarsi come una smart card in questa modalità. Una smart card è una carta tascabile con dei circuiti integrati (solitamente memoria volatile e microprocessore). In questa modalità un lettore RFID esterno non può effettuare distinzioni tra una carta e un dispositivo NFC. Una smart card può essere emulata sia a livello applicativo sia usando un componente chiamato Secure Element, un dispositivo simile alle vere smart card. Ovviamente questa modalità è utile per i pagamenti elettronici.

- *Modalità peer to peer*: questa modalità permette a due dispositivi NFC di stabilire una connessione bidirezionale per scambiare dati. Si distingue qui l'initiator, cioè chi fa la richiesta (client) e il target, cioè

chi riceve e risponde alla richiesta (host). L'host non può iniziare la comunicazione se prima non è stato interpellato dall'initiator. Eventuali misure di sicurezza devono essere implementate dallo sviluppatore al livello aplicativo, perchè gli standard per la comunicazione non speficicano niente da questo punto di vista. Per quanto riguarda i consumi energetici, l'energia richiesta per creare il campo NFC è condivisa tra il targer e l'initiatior.

2.4.1 Standard e specifiche

Per garantire l'interoperabilità tra dispositivi di tipo diverso, anche per questa tecnologia sono necessari standard e specifiche. Lo standard fondazionale è ISO/IEC 18092, "Near Field Communication-Interface and Protocol", che è stato approvato per la prima volta nel 2003, poi revisionato recentemente nel 2013. Questo standard definisce l'interfaccia e il protocollo per la comunicazione tra due dispositivi vicini, in modo da appunto garantire l'interoperabilità tra iniziatore, destinatario e tra le applicazioni sviluppate da diverse aziende. Questo standard è stato costruito attorno ai già esistenti standard RFID, includendo ISO/IEC 14443 e JIS 6319-4, che operano a 13.56 MHz. Un secondo standard NFC, ISO/IEC 21481,"Near Field Communication Interface and Protocol-2" è stato approvato per la prima volta nel 2005 e è stato revisionato nel 2012. Questo standard è stato sviluppato per determinare quale dei protocolli (ISO/IEC 18092, ISO/IEC 14443, ISO/IEC 15693) deve essere usato per uno specifico scambio dei dati, creando un ponte tra gli standard già esistenti. ISO/IEC 15693, standard per la comunicazione senza fili, è stato incluso per aumentare l'interoperabilità tra la nuova tecnologia NFC e i sistemi RFID già presenti. Atri standard e specifiche sono sviluppati e mantenuti dall'associazione noprofit NFC Forum (http://nfc-forum.org), fondata da Sony, Philips e Nokia nel 2004.

2.4.2 Architettura di NFC

E' necessario introdurre una possibile architettura di NFC, in modo da mostrare i componenti coinvolti e le loro interconnessioni, come mostrato in figura 2.9. I principali componenti sono:

- *Host controller*: l'ambiente dove risiede l'applicazione, ad esempio lo smartphone. L'applicazione può essere presente nella SIM Card, nel

Secure Element o nella smart card (componenti contenuti nell'host controller).

- *Secure element*: ambiente sicuro dove dati importanti come i dati della carta di credito sono salvati. Si trova all'interno dell'host controller. Affinchè sia sicuro, il tutto deve essere resistente alle manipolazioni,deve essere in grado di eseguire funzioni di crittografia e deve poter eseguire software per la sicurezza.

- *NFC Controller*: è il collegamento tra l'host controller e il secure element. Ci sono vari protocolli tra l'host controller e l'NFC controller, come SPI (Serial Peripheral Interface) e USB (Universal Serial Bus). Per la comunicazione con il secure element c'è il protocollo single wire e l'interfaccia NFC wired.

- *NFC Antenna*: insieme di fili che occupa il più spazio possibile nel dispositivo.

- *Baseband controller*: svolge le funzioni radio.

- *RF Unit*: prende in ingresso i segnali provenienti dal Baseband controller e li modula.

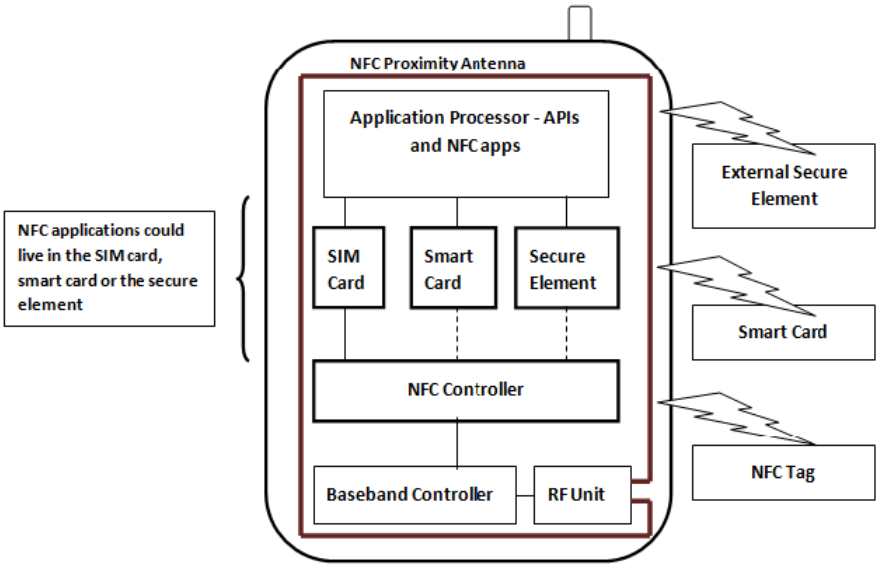

Figura 2.9: Architettura NFC di uno smartphone

Capitolo 3

Middleware per Internet of Things

3.1 Introduzione

Il concetto di middleware è piuttosto ampio e non è facile darne una definizione precisa, poichè questo termine viene usato per indicare diverse tipologie di software. Tuttavia, come già accennato precedentemente, un middleware è una piattaforma software che risiede tra il sistema operativo e le applicazioni. Sono diverse le ragioni per cui sviluppare questa piattaforma risulta necessario, e quella che spicca sopra le altre è la necessità di gestire allo stesso modo dispositivi di tipo diverso. I tipi di hardware e software in gioco sono numerosissimi, e senza un modo comune di gestirli, creare un sistema risulta difficoltoso. Il middleware può essere anche anche una tecnologia di integrazione, perchè può essere progettato per fare comunicare tra di loro sistemi e applicazioni già esistenti di tipo diverso, fungendo da colla. Oltre a tutto questo, deve fornire alle applicazioni API e servizi, per migliorare la qualità del servizio. In linea di massima, un middleware per IoT deve trattare le seguenti tematiche:

- *Interoperabilità*: si ha spesso a che fare in IoT con la necessità di far condividere informazioni tra dispositivi (e applicazioni) di tipo diverso, che devono cooperare a uno scopo comune. L'interoperabilità si può suddividere in tre tipologie:

  - *Di rete*: l'interoperabilità di rete riguarda protocolli per lo scambio di informazioni attraverso le reti, senza però pensare al contenuto delle informazioni. riguarda i problemi di connettività di base reltivi ai livelli dello stack TCP-IP (in questo caso livello fisico, data link, di trasporto, sessione e di applicazione).
  - *Sintattica*: riguarda il formato e la struttura della codifica delle informazioni scambiate (in questo caso riguarda il livello di presentazione e di applicazione dello stack TCP-IP).
  - *Semantica*: riguarda le regole per interpretare le informazioni.

  Un middleware deve fornire API per soddisfare le problematiche relative all'interoperabilità.

- *Consapevolezza del contesto*: il middleware deve essere consapevole del contesto in cui opera, in modo tale da fornire all'utente le informazioni necessarie nel momento in cui ne ha bisogno. Si può raggiungere questo obiettivo in due passaggi:

  - *Rilevazione del contesto*: vengono raccolti i dati e identificati i fattori significativi che possono incidere nella risposta da fornire all'utente.
  - *Elaborazione del contesto*: il tutto viene elaborato e viene scelta la risposta da fornire in base al contesto.

  Per implementare questi due passaggi si può procedere ad esempio in questo modo: ogni servizio che il middleware supporta si registra con un identificatore che dipende dal contesto. Questo identificatore viene cambiato in base ai cambiamenti di contesto (ad esempio quando si cambia locazione).

- *Gestione e scoperta di dispositivi*: i dispositivi in gioco devono poter vedere la presenza degli altri intorno a loro, e viceversa, in modo tale da poter cooperare insieme.

- *Privacy e sicurezza*: dato che in gioco c'è la raccolta di dati, questi devono essere protetti a dovere. Un modo per realizzare questo è utilizzare reti peer to peer sicure, oppure implementare dei sistemi di autenticazione dei dati, gestione di permessi e altro.

- *Gestione dei dati*: i dati raccolti in gioco sono innumerevoli, e di ogni tipo. Questi vanno quindi gestiti, filtrati e salvati in modo opportuno.

A questi requisiti se ne possono poi aggiungere altri, in base al tipo di sistema da realizzare e ai suoi requisiti (ad esempio affrontare la questione dei consumi energetici e delle risorse limitate). Successivamente verrà introdotta una possibile architettura per IoT, e infine analizzate le tematiche che devono affrontare i middleware dedicati a sistemi di reti di sensori e a sistemi RFID, con alcuni esempi.

3.2 Architettura orientata ai servizi per IoT

Per integrare sistemi e dispositivi di tipo diverso, una architettura orientata ai servizi (SOA) è una possibile scelta giusta. Questo tipo di architettura viene utilizzata già in settori come il cloud computing. Sono state proposte alcune idee di architetture SOA multilivello per IoT. Per avere una visione generale, verrà mostrata una architettura a quattro livelli: sensing, networking, service, interface. Nella progettazione di questa architettura vanno tenuti in considerazione requisiti come l'estendibilità, la scalabilità, la modularità e l'interoperabilità tra dispositivi diversi. Considerando anche che gli oggetti possono muoversi o anche aver bisogno ad esempio di interagire in tempo reale con l'ambiente, è anche necessario che l'architettura sia adattativa. La figura 3.1 mostra l'organizzazione dei livelli, dei quali verrà specificato il ruolo.

- *Sensing*: in questo livello, i sistemi basati su tag RFID, sensori e altri dispositivi possono fare rilevazioni sull'ambiente circostante e scambiare informazioni.

- *Networking*: il ruolo di questo livello è di connettere gli oggetti tra di loro e di permettere di condividere le informazioni raccolte con gli altri. Inoltre, questo livello può aggregare le informazioni provenienti da altre infrastrutture (come sistemi aziendali, sistemi di trasporto, reti elettriche, sistemi sanitari e altro). Questo livello deve fornire il supporto base alla rete e al trasporto di dati attraverso reti senza fili o cablate. Durante il progetto di questo livello, i progettisti devono far fronte a problemi come la gestione della rete per dispositivi eterogenei, efficienza di consumi, qualità del servizio, sicurezza etc.

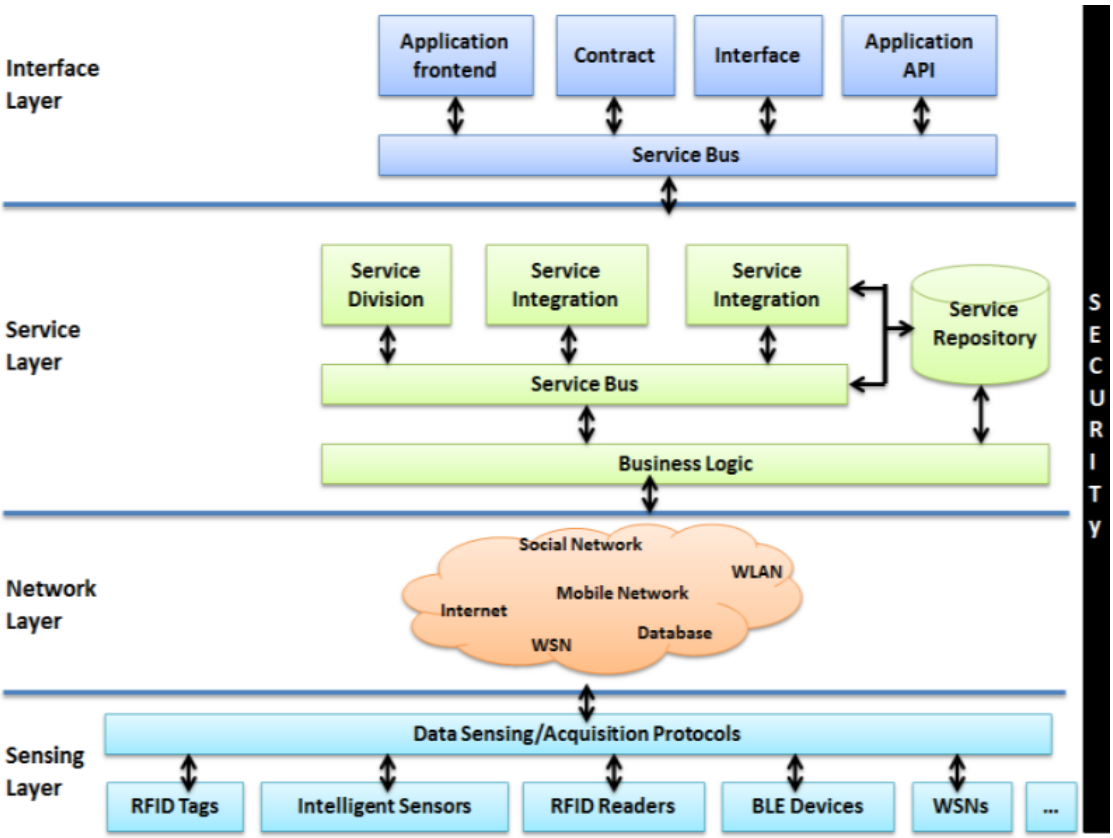

Figura 3.1: Architettura SOA di IoT

- *Service*: questo livello è basato sul middleware, che fornisce funzionalità per integrare senza problemi servizi e applicazioni in IoT. Il middleware è anche una piattaforma che permette di ridurre i costi, poichè permette il riuso di componenti software e hardware, fornendo uno strato tra il sistema operativo e le applicazioni, garantendo l'interoperabilità. Il service layer deve essere in grado di identificare i requisiti applicativi e di fornire API e protocolli per supportare i servizi richiesti (sviluppati da diverse organizzazioni), in base ai bisogni degli utenti. Qui si gestiscono anche tutti i problemi relativi ai servizi come lo scambio e la memorizzazione di informazioni. Il livello contiene i seguenti componenti:
  - *Service discovery*: necessario per trovare oggetti che possono offrire un certo servizio o informazione in maniera efficiente
  - *Service composition*: permette l'interazione e la comunicazione tra oggetti connessi. Il componente precedente sfrutta le relazioni tra i differenti oggetti per trovare il servizio desiderato, mentre questo componente schedula o ricrea servizi più adatti, in modo da ottenere il servizio che soddisfi la richiesta in modo migliore.
  - *Trustworthiness management*: fornisce meccanismi che permettono di utilizzare le informazioni fornite dai servizi in modo sicuro.
  - *Api per i servizi*: necessarie per supportare l'interazione tra i servizi.
- *Interface*: questo livello fornisce meccanismi per l'interazione col sistema agli utenti e anche meccanismi per le applicazioni. Spesso i dispositivi in gioco sono prodotti da aziende diverse, e questi non utilizzano sempre gli stessi protocolli e standard. Inoltre, sono frequenti scenari in cui dispositivi vengono aggiunti dopo che i sistemi sono stati creati. E' quindi difficile fare in modo che nuovi dispositivi si connettano dinamicamente e riescano a funzionare col sistema già presente. Serve un livello (interface) per semplificare la gestione e l'interconnessione degli oggetti distribuiti nella rete. A tal scopo, si usano IFP (Interface Profile), ovvero sottogruppi di standard per permettere l'interazione tra applicazioni distribuite. Vengono usati per descrivere le specifiche tra applicazioni e servizi.

3.3 Middleware per Wireless Sensor Network

Come detto, a causa della grande eterogeneità di hardware, sistemi operativi, protocolli di rete, il middleware deve essere in grado di fornire un ambiente distribuito per i servizi di comunicazione. Inoltre il gap tra i requisiti delle applicazioni di alto livello e il livello fisico è troppo ampio. Deve anche fornire, facendo da ponte tra i sistemi operativi e le applicazioni, migliore qualità del servizio, sicurezza e migliore gestione delle risorse. Può fornire un ambiente per il supporto e la coordinazione di più applicazioni. Gli sviluppatori di applicazioni per le reti di sensori devono far fronte a tantissimi vincoli per integrare i nodi sensori col mondo fisico, quindi progettare un middleware diventa fondamentale. In particolare, il middleware deve affrontare le seguenti problematiche:

- *Gestione delle risorse*: conviene effettuare la gestione delle risorse a livello di middleware, piuttosto che a livello di sistema operativo o a livello applicativo. Se venisse fatto a livello di sistema operativo la gestione delle risorse sarebbe dipendente dalla piattaforma, e se lo facessimo a livello applicativo non sarebbe comune a tutte le applicazioni. Conviene farlo una volta sola a livello di middleware.

- *Data centrismo*: la comunicazione nelle reti di sensori è data centrica, perchè siamo interessati ai dati raccolti, piuttosto che ai nodi sensori in sè. Il middleware deve quindi supportare la comunicazione data centrica.

- *Fusione dei dati*: spesso in gioco ci sono tantissimi nodi sensore. Inoltre, come detto nella sezione precedente, spesso non è appropriato che un nodo sensore trasmetta direttamente i dati alla stazione base. I dati trasmessi poi sono spesso ridondanti, sprecando banda e energia. Il processo di fusione dei dati unisce copie di dati e informazioni per una maggiore efficienza.

- *Gestione dei consumi*: anche se in certi casi ci sono delle batterie per l'alimentazione dei nodi sensori, spesso queste non sono sostituibili. E' quindi evidente che limitare l'uso delle risorse è fondamentale ed è uno dei principali problemi, e può essere contenuto con la gestione dinamica della potenza e con il ridimensionamento dinamico dei voltaggi. Il middleware deve essere leggero e deve essere in grado di gestire al

meglio le risorse dei dispositivi in modo da allungare il più possibile il loro ciclo di vita. Questo avrebbe anche impatto ovviamente sulle spese future.

- *Scalabilità e tolleranza ai guasti*: a causa del grande numeri di nodi, e del fatto che alcuni nodi possono anche essere mossi, i fallimenti non sono poco frequenti e possono incidere sulla scalabilità, modificando la topologia della rete. Un middleware adeguato può aiutare a migliorare la tolleranza ai guasti del sistema, oltre che essere in grado di scalare su e giù.

- *In-network processing*: per in-network processing si intende il permettere ai nodi sensore di prendere decisioni sul come gestire la rete in base ai dati che hanno trasmesso (ad esempio aggregazione, compressione, codifica di dati). Il middleware deve fornire meccanismi per supportare in network processing per garantire un buon livello di prestazioni anche se la rete cresce di dimensioni in futuro.

- *Adattamento dinamico*: la topologia della rete di sensori può cambiare di frequente, risulta quindi necessario che il middleware debba supportare il cambiamento dinamico della rete e adattarsi per supportarla.

- *Eterogeneità*: un middleware dovrebbe fornire modelli di programmazione per ridurre il gap tra l'hardware e le applicazioni. Il middleware deve supportare i dispositivi che si interfacciano con i vari tipi di hardware e reti. Inoltre con il middleware è possibile integrare una rete di sensori con altre reti.

- *Conoscenza delle applicazioni*: il middleware deve essere progettato per supportare una vasta gamma di applicazioni diverse, in numero più alto possibile.

- *Sicurezza*: spesso le reti di sensori sono soggette ad infiltrazioni e ad attacchi: il middleware deve fornire quindi meccanismi per migliorare la sicurezza della rete. Come nel caso della gestione delle risorse, conviene gestire la sicurezza a livello di middleware, e non a livello di sistema operativo o applicativo.

- *Qualità del servizio*: parametri di misurazione della qualità del servizio sono la larghezza di banda, tasso di perdita dei dati, reattività etc. Il middleware deve quindi tenere conto di questi aspetti, per garantire un servizio affidabile.

I middleware possono essere classificati in base al tipo di programmazione e all'approccio usato.

- *Approccio con macchina virtuale*: approccio flessibile, che contiene macchine virtuali, interpreti e agenti mobili. Permette agli sviluppatori di scrivere applicazioni in piccoli moduli separati. Il sistema distribuisce i moduli attraverso la rete con degli algoritmi, minimizzando i consumi di energia e l'uso delle risorse. La macchina virtuale quindi interpreta i moduli.

- *Approccio con programmazione modulare*: le applicazioni sono divise in programmi modulari per facilitarne la distribuzione attraverso la rete. Trasmettere singoli moduli richiede meno consumo energetico che trasmettere un'intera applicazione. Ma questo approccio non permette eterogeneità di hardware.

- *Approccio con base di dati*: la rete di sensori è trattata come un database distribuito. Possiede una interfaccia semplice da usare, e usa query simili a quelle SQL per gestire i dati. Non possiede però il supporto per applicazioni in tempo reale, fornendo così risultati approssimativi.

- *Approccio application driven*: questo approccio permette di ottimizzare la rete e di regolare i consumi e l'uso di risorse in base ai bisogni delle applicazioni. Le applicazioni specificano una certa qualità del servizio richiesta, e il middleware si adatta di conseguenza (ad esempio usando diverse combinazioni di nodi sensore).

- *Approccio orientato ai messaggi*: i middleware orientati ai messaggi usano il pattern publish/subscribe (generalmente trasmettitori e ricevitori di messaggi dialogano attraverso un dispatcher o broker, componente software che fa da intermediario. Il trasmettitore di un messaggio non deve essere conoscere il ricevitore,ma si limita a consegnare il messaggio al dispatcher, che provvederà poi a inviarlo al destinatario). Supporta la comunicazione asincrona, permettendo il disaccoppiamento tra il trasmettitore e il ricevitore.

Nelle sezioni successive verranno mostrati degli esempi di middleware sviluppati per le reti di sensori. Lo scopo della tesi non è introdurre concetti tecnici e approfondire questi middleware, ma solo quello di mostrare degli esempi relativi agli approcci descritti sopra. Come caso di studio più approfondito verrà analizzato Java Embedded, middleware costruito con un approccio basato su macchina virtuale.

3.3.1 Esempi di approccio basato su macchina virtuale

Di seguito vengono elencati alcuni middleware basati su macchina virtuale:

- *Mate*: Questo middleware gira sul sistema operativo TinyOS, sviluppato appositamente per le reti di sensori. Questo sistema utilizza meccanismi come interruzioni e routine ad esse associate, perciò si può dire che sia basato sugli eventi. Mate è sostanzialmente un interprete di bytecode. Ci sono 24 pacchetti di istruzioni a disposizione. Si concentra in particolare sui limiti energetici delle reti, fornendo inoltre meccanismi per l'interazione e l'adattabilità dei componenti. Fornisce agli utenti un'interfaccia di alto livello per accedere ai dati dei sensori. La gestione dell'invio e ricezione dei pacchetti viene fatta in modo sincrono.

- *MagnetOS*: MagnetOS in realtà è un sistema operativo, ma include anche le funzioni di un middleware. E' un sistema distribuito adattativo, scalabile e attento ai consumi. E' ovviamente costruito in modo tale da adattarsi ai limiti di risorse dei dispositivi a cui è rivolto. Si adatta in particolar modo ai cambiamenti della topologia della rete, adattandosi all'evenienza, facendo in modo che i programmatori non debbano intervenire in caso di cambiamenti di hardware.

- *Scalable Wireless Sensor Network Query Machine*: SwissQM è una macchina virtuale a livelli per le reti di sensori wireless. Definisce un set di istruzioni bytecode indipendenti dalle tipologie di sensori e dai linguaggi di programmazione. Questo porta a numerosi vantaggi, perchè spesso i programmatori devono utilizzare linguaggi di basso livello ad hoc per il tipo di sensore o dispositivo che necessitano della conoscenza di molti dettagli. Un sistema quindi traduce le istruzioni

dei vari linguaggi di programmazione nel bytecode che poi la macchina virtuale esegue.

3.3.2 Esempi di approccio basato su basi di dati

Di seguito vengono elencati alcuni middleware basati su basi di dati:

- *Cougar*: Cougar utilizza l'approccio database. Viene utilizzato un linguaggio SQL per effettuare le operazioni sui sensori (ad esempio per comandare la raccolta di dati), e i dati raccolti salvati in un database. Ha il vantaggio di limitare i consumi, perchè solo lato utente si richiedono le varie operazioni, quindi i dati non vengono salvati dai sensori localmente.

- *Sensor Information Networking Architecture (SINA)*: SINA vede i sensori come insiemi di attributi, rappresentanti i dati da essi raccolti. Ogni nodo sensore mantiene una tabella con i dati che raccoglie. Come nel caso precedente, usando delle query è possibile recuperare i dati.

- *TikiriDB*: TikiriDB è un livello del sistema operativo Contiki, rappresentante un database. Fornisce funzionalità per permettere di recuperare dati dalle reti di sensori basate su questo sistema operativo. Per recuperare i dati si usa un linguaggio a query. Il funzionamento è il seguente: l'utente effettua delle query, a seguito delle quali vengono generati dei pacchetti. TikiriDB li riceve e invia i pacchetti di risposta all'utente.

- *TinyDB*: TinyDB è un sistema distribuito che gira su TinyOS, e utilizza un linguaggio a query per recuperare i dati dai sensori. Il tutto tenendo conto ovviamente di problemi relativi ai consumi e alle limitate risorse a disposizione.

- *TinyLIME*: TinyLIME è un middleware basato su un approccio a databse che gira su TinyOS, ma è basato su LIME(a sua volta un middleware), che viene esteso aggiungendo le funzionalità necessarie per le reti di sensori. Gli utenti possono recuperare dati dai sensori con delle query. Il tutto però ha molte limitazioni, poichè è possibile solo raccogliere dati da sensori locali, poichè non è supportata la propagazione della query.

- *Not Only SQL (NoSQL)*: Il concetto di NoSQL è diverso da quello relativo ai database relazionali. In quest'ultimi le query devono rispettare le proprietà ACID (atomicità, consistenza, isolamento e durabilità). In NoSQL questo non è garantito, si tiene solo conto di performance e scalabilità. Quindi è indicato per sistemi che hanno a che fare con grandi moli di dati, perchè è più performante. Si possono quindi sfruttare database e middleware basati su questo modello per le reti di sensori.

3.3.3 Esempi di approccio modulare

Di seguito vengono elencati alcuni middleware costruiti con approccio modulare:

- *Impala*: Impala utilizza l'approccio a programmazione modulare. E' orientato alla gestione automatica dei cambiamenti della rete e al supporto di applicazioni dinamiche. Il middleware è diviso in due livelli. Il livello superiore contiene le applicazioni e i protocolli. Il livello inferiore contiene tre agenti:

  - *Event filter*: si occupa di ricevere e propagare eventi per i dispositivi, in modo da favorirne l'interazione.
  - *Adapter*: gestisce l'adattabilità delle applicazioni sulla base dei possibili diversi scenari, tenendo conto di fattori come l'efficienza energetica.
  - *Updater*: si occupa di gestire gli aggiornamenti software dei componenti.

- *Agilla*: Agilla è un middleware ad agenti mobili che gira su TinyOS. Possiede un'architettura a livelli, in modo tale da diminuire la dimensione del codice. Gli agenti possono spostarsi da un nodo all'altro con due istruzioni: clone e move. Su ogni nodo possono girare al massimo quattro agenti, permettendo così di poter utilizzare più applicazioni su un singolo nodo.

3.3.4 Esempi di approccio application driven e a messaggi

Di seguito verranno mostrati un esempio relativo all'approccio application driven (MILAN) e uno relativo all'approccio a messaggi (Mires):

- *Middleware Linking Application and Networks (MILAN)*: MILAN permette alle applicazioni dedicate alle reti di sensori di specificare i loro requisiti di qualità, adattando quindi le caratteristiche della rete per soddisfarli. Le applicazioni possono inviare i propri requisiti a MILAN continuamente nel tempo.

- *Mires*: Mires utilizza un modello di comunicazione asincrono con un meccanismo di publish-subscribe. Anche Mires gira su TinyOS, e risulta particolarmente adatto grazie alla natura basata su scambio di messaggi e basata su eventi di questo sistema operativo.

3.4 Middleware per sistemi RFID

La raccolta di dati e la loro elaborazione nei sistemi RFID richiede delle soluzioni software complesse. La complessità cresce anche col passare del tempo, perchè gli affari e gli scenari applicativi crescono e si diversificano. Quindi, se il middleware è importante in diversi ambiti (cloud computing, reti wireless etc) lo è anche nei sistemi RFID. Solitamente in questi sistemi il middleware si trova tra il lettore (interrogatore) e le applicazioni. Il suo obiettivo principale è quello di raccogliere ingenti quantità di dati grezzi provenienti dai tag RFID, filtrarli, compilarli in un formato utilizzabile e di distribuirli ai sistemi informativi. Si utilizza quando è necessario condividere dati con più di una entità allo stesso tempo, come nelle catene di fornitura dove più lettori sono distribuiti lungo industrie, magazzini e centri di distribuzione. Generalmente i middleware per i sistemi RFID sono costituiti da tre livelli, come mostrato in figura 3.2 Ogni livello ha il suo preciso compito:

- *Service Management*: questo livello collega il sistema al mondo reale. Il suo compito principale è quello di ricevere le richieste dai sistemi informativi (o altri middleware) e filtrare i risultati a partire dai dati recuperati dai lettori.

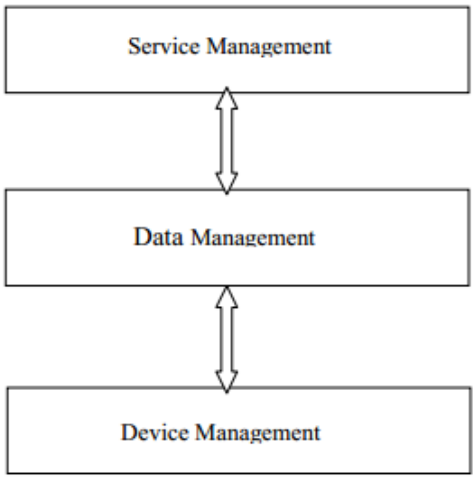

Figura 3.2: Architettura del Middleware in sistemi RFID

- *Data Management*: questo livello raccoglie i dati grezzi, li filtra, li pulisce e li converte in un formato che può essere meglio elaborato.
- *Device Management*: questo livello deve gestire e monitorare i lettori sparsi nel sistema.

Concludendo, il progetto di un middleware per sistemi RFID deve essere in grado di far fronte a problemi di affidabilità, scalabilità, bilanciamento del carico e gestione dei dati.

3.4.1 Alcuni esempi

Di seguito verranno mostrati alcuni esempi di esistenti middleware per sistemi RFID. Come detto per quanto riguarda quelli basati su sistemi wireless, vengono introddotti solo allo scopo esemplificativo, e non vogliono andare nello specifico.

- *Multi-agent Based RFID Middleware*: come suggerisce il nome, questo middleware sfrutta il paradigma di programmazione ad agenti. E' composto dai tre livelli mostrati nella sezione precedente. Le applicazioni specificano i dati che vorrebbero ricevere. Gli agenti a turno

comunicano con gli agenti dei lettori nel livello device, recuperano i dati grezzi, li elaborano e inviano la risposta.

- *WinRFID*: è basato su una architettura di cinque livelli: hardware, protocollo, data processing, framework XML e data presentation. Il livello hardware si occupa dei problemi relativi alla comunicazione tra i tag e i lettori e la loro configurazione. Il livello di protocollo è un'astrazione dei principali protocolli e standard di lettura per i lettori. Dopo la raccolta dei dati grezzi, il livello data processing si occupa di elaborarli. Questi dati vengono inviati al livello di framework XML che li sistema in formato XML. Infine il livello data presentation si occupa di utilizzare i dati provenienti dal livello precedente per gli scopi applicativi.

- *RF2ID - Reliable Framework for RFID*: framework sviluppato per questioni come affidabilità, bilanciamento del carico, scalabilità e organizzazione dei dati. L'architettura di questo middleware è basata sul concetto del virtual reader (VR) e del virtual path (VP). Ogni VR è associato a un insieme di lettori fisici nella sua regione geografica, e si occupa di filtrare i dati provenienti da essi e di gestirli. I VP sono canali logici creati dinamicamente dai VR per fornire le funzionalità elencate precedentemente.

- *LIT Middleware - Logistic Information Technology*: sviluppato sul concetto di Application Level Events (ALE) e di EPC Information Services (EPCIS)(sono degli standard di EPCglobal). Un'applicazione basata su ALE deve descrivere le possibili operazioni che potrebbe eseguire su un tag, e ALE trova il modo più appropriato di soddisfare queste richieste. EPCIS definisce le modalità di condivisione dei dati.

3.5 Altri esempi

Altri esempi importanti di middleware per IoT, questa volta non mirati a una tecnologia in particolare come nei casi precedenti, possono essere trovati in Mbed e Java Embedded (quest'ultimo verrà approfondito nel capitolo 4).

- *Mbed*: piattaforma orientata specificatamente per i microcontrollori ARM Cortex-M. E' ottimizzata quindi per dispositivi rivolti a IoT, cioè con risorse limitate. La piattaforma offre:

  - *Sistema operativo*: sistema operativo che viene installato sui dispositivi in questione. Include servizi e API necessarie per gestire al meglio dispositivi di IoT (connessione, gestione delle risorse, sicurezza etc). Il sistema operativo include un framework scritto in C++ per poter creare applicazioni per i dispositivi, che serve per permettere di non curarsi di aspetti di basso livello durante la programmazione di microcontrollori. Inoltre garantisce interoperabilità, poichè le applicazioni possono essere riusate per microcontrollori diversi.
  - *Device Server*: gestisce le connessioni dei dispositivi, la sicurezza e la comunicazione,
  - *Tools*: vengono forniti ovviamente anche dei tool per lo sviluppo. Tra questi ci sono dei tool a linea di comando e tool di test, che insieme formano un toolkit che permette di facilitare lo sviluppo di applicazioni IoT. Viene fornito anche un ambiente di sviluppo cloud accessibile via internet, in modo che sia accessibile da dove si vuole. Essendo cloud, non sarà necessario aggiornarlo poichè viene aggiornato lato server.

- *Java Embedded*: Un'altra piattaforma importante, che sarà oggetto del caso di studio del capitolo successivo (quindi qui sarà solo introdotta) è Java Embedded di Oracle. La piattaforma è composta da una macchina virtuale (la Java Virtual Machine) che va installata sui dispositivi coinvolti nel sistema da realizzare. In questo modo è possibile programmare su dispositivi diversi usando lo stesso linguaggio di programmazione, garantendo così anche la portabilità delle applicazioni e semplificando lo sviluppo. Vengono messe anche a disposizione numerose API per gestire periferiche, connessione e altro, in modo da non doversi preoccupare di aspetti di basso livello e di hardware durante lo sviluppo. Vengono anche messi a disposizione l'ambiente di sviluppo e altri tool per il testing delle applicazioni.

Capitolo 4

Caso di studio: Java Embedded

Java Embedded ME (Micro Edition) 8 è stato introdotto nel marzo del 2014, allineato alla SE (Standard Edition) 8 per quanto riguarda la macchina virtuale, il linguaggio di programmazione e le librerie, rendendo potenzialmente in grado a chi già ha familiarità con la SE di potere sviluppare applicazioni relative a IoT. L'importanza di IoT di questi tempi è messa in risalto dal fatto che Oracle abbia deciso di rilasciare due versioni diverse della sua piattaforma. Le versioni sono comunque allineate e simili, dato che i modelli di programmazione usati non cambiano, ma ME 8 è rivolta a IoT e ai relativi dispositivi in gioco(come quelli descritti nel capitolo due), e fornisce una piattaforma ottimizzata per loro. I dispositivi target infatti hanno meno di 128 KB di memoria RAM e 1 MB di memoria ROM/Flash. Sono state aggiunte anche nuove API e nuove caratteristiche relative a un ambiente embedded rispetto alla SE. In una normale architettura, le applicazioni si scrivono in base al sistema operativo che c'è sotto. Se usiamo la piattaforma Java (con la sua filisofia "Write once, run anywhere"), possiamo creare applicazioni a prescindere dal sistema operativo, grazie alla macchina virtuale. Questo è un enorme vantaggio, poichè fa fronte al problema della grande eterogeneità dei dispositivi in gioco (sia per quanto riguarda l'hardware sia per quanto riguarda il sistema operativo). La piattaforma non permette solo questo, ma fornisce anche servizi alle applicazioni, grazie alle innumerevoli API disponibili. Questa piattaforma permette quindi un risparmio considerevole in termini di tempo e di denaro rispetto a soluzioni

ad hoc, non solo in termini di sviluppo, ma anche in termini di manutenzioni future e di portabilità. Un aspetto da sottolineare è la disponibilità di tantissime API per accedere alle periferiche di I/O, permettendo così di accedere a sensori e altri dispositivi.

4.1 La macchina virtuale

La piattaforma Java si basa sulla sua macchina virtuale, la Java Virtual Machine (JVM). Questo permette di poter usare lo stesso linguaggio di programmazione con hardware di tipo diverso. La macchina virtuale si pone tra il sistema operativo e l'applicazione, come mostrato in figura 4.1, facendone da intermediaria. Questo semplifica anche la portabilità delle applicazioni. Per sviluppare applicazioni, si utilizza il linguaggio di programmazione Java (linguaggio orientato agli oggetti). In figura 4.2 è mostrata l'architettura della macchina virtuale. I suoi componenti sono:

- *Classloader*: sottosistema della macchina virtuale che si occupa di trovare e eseguire i file .class derivati dalla compilazione dei file .java. I passi che esegue sono:
  - *Loading*: trova e importa i dati binari per un tipo
  - *Linking*: verifica la correttezza del tipo importato e alloca memoria per le variabili
  - *Initialization*: inizializza le variabili al loro valore iniziale
- *Runtime data areas*: quando una macchina virtuale esegue un programma, ha bisogno di salvare bytecode, oggetti, parametri e tanto altro. Per questo la JVM organizza la memoria in diverse aree. Ogni istanza della JVM ha una method area e heap (condivise da tutti i thread in azione). Quando esegue un file .class, il class loader salva i tipi caricati nella method area. Quando il programma è in esecuzione, gli oggetti istanziati vengono messi nello heap. I thread possiedono uno stack e un registro. Il registro memorizza la prossima istruzione da eseguire. Nello stack invece vengono salvate le variabili, i parametri dei metodi e altro. E' presente anche un'area per i dati nativi, metodi che possono sostanzialmente accedere a qualunque area di memoria della JVM senza restrizioni.

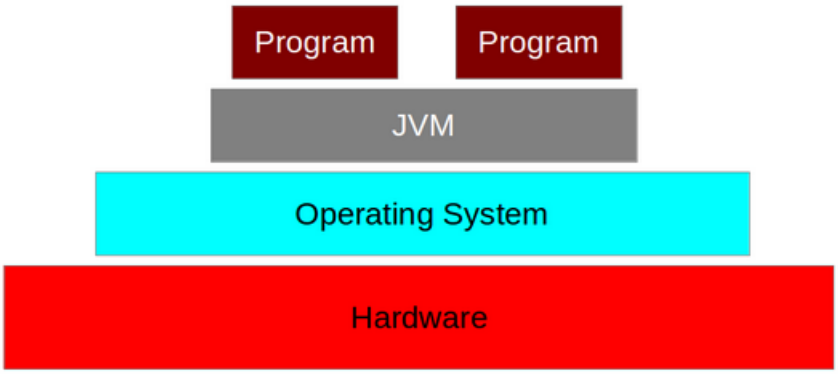

Figura 4.1: Posizione della Java Virtual Machine

- *Execution engine*: si occupa di eseguire il bytecode. Con le specifiche degli instruction set, l'engine saprà esattamente cosa fare quando incontrerà le varie istruzioni.

- *Native method interface e native method libraries* : si occupano della gestione di eventuali metodi nativi. Serve nel caso che sia necessario eseguire del codice macchina che altrimenti non sarebbe rappresentabile in linguaggio Java.

Quando un'applicazione Java viene lanciata, viene creata un'istanza della macchina virtuale (un'istanza per ogni applicazione).

4.2 Architettura della piattaforma

La piattaforma è un middleware, poichè si pone tra il sistema operativo e le applicazioni facendo da intermediario, offrendo anche servizi alle applicazioni stesse. In figura 4.3 è mostrata l'architettura della piattaforma Java ME.

- *Java VM*: questo livello è la macchina virtuale che si piazza sopra al sistema operativo permettendo di non dipendere più da esso, e contiene il Java ME Connected Limited Device Configuration (CLDC)

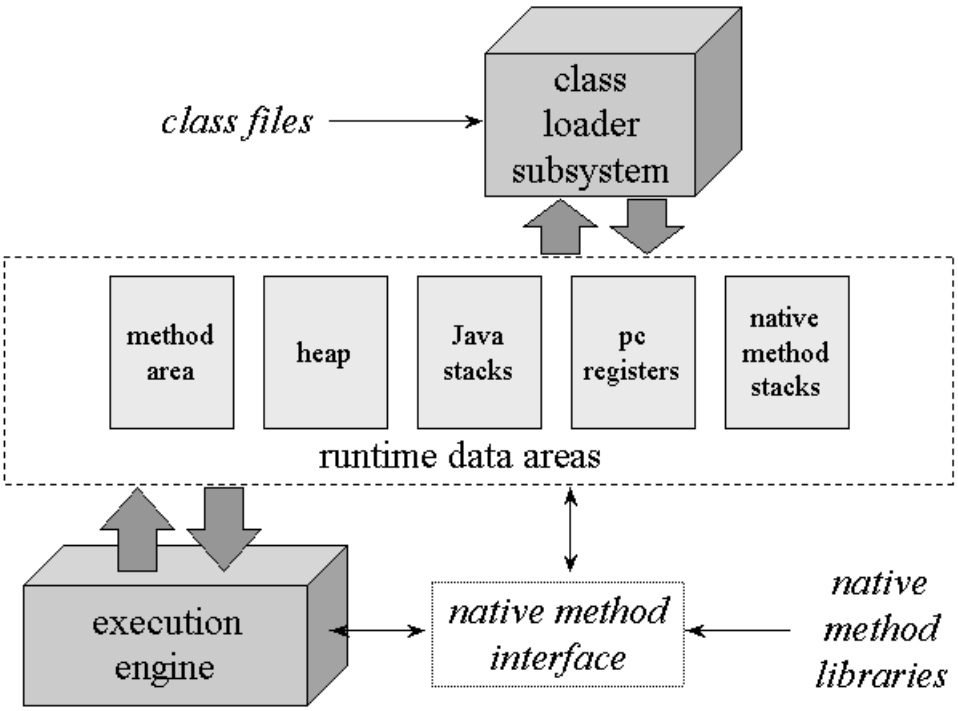

Figura 4.2: Architettura della Java Virtual Machine

(framework che contiene delle API e delle librerie di base necessarie per le applicazioni)

- il livello sopra la macchina virtuale contiene:
  - *Device IO Access*: varie API per connettersi alle periferiche. Sono supportati ad esempio:
    - *SPI*: accesso agli slave SPI (come dispositivi audio, display, memorie EEPROM/Flash..)
    - *I2C*: accesso agli slave I2C (come sensori e altri dispositivi)
    - *GPIO*: accesso ai pin GPIO (quindi a bottoni e led ad esempio)
    - *Conversione analogico/digitale*: accesso a convertitori analogico-digitale e digitale-analogico
    - *UART*: accesso alle porte seriali UART

    Per quanto riguarda le periferiche, è possibile usare il Device Abstraction Interface: grazie alla classe DeviceManager è possibile determinare il tipo di periferica a tempo di esecuzione, facilitando molto la programmazione poichè permette di trattare in modo simile periferiche di tipo diverso. Si può quindi infine pensare al seguente scenario: una board come Raspberry Pi o Arduino, con dei sensori, attuatori o altri dispositivi connessi ai pin. Tramite le API messe a disposizione, è possibile recuperare dati da essi o operare su di loro.
  - *Generic Connection Framework (GCF)*: serie di API per permettere alle applicazioni di controllare la connettività (senza fili o con fili). I tipi di connessione supportati sono tanti, e per una lista si faccia riferimento alla figura 4.4. Da sottolineare anche il supporto a IPv6.
  - *Java ME Embedded Profile (MEEP)*: orientato a dispositivi medio piccoli. E' costruita sopra il CLDC, e possiede API e librerie necessarie per fornire un ambiente adatto a dispositivi con capacità limitate.
  - *Security and Trust Services API (SATSA)*: fornisce servizi per la sicurezza, come meccanismi per la crittografia, meccanismi di autenticazione e altro.

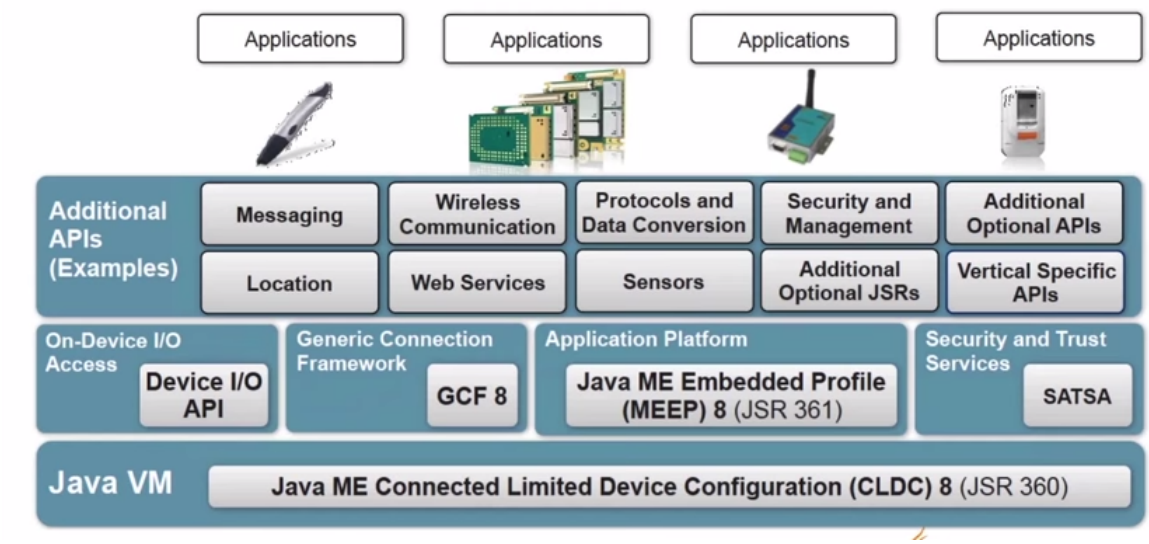

Figura 4.3: Architettura di Java ME

- *Varie API*: contiene API aggiuntive per i vari ambiti, con varie librerie opzionali a disposizione, come indicato nella figura.

Infine in cima si trovano ovviamente le applicazioni. Oracle fornisce i seguenti tool per lo sviluppo di applicazioni:

- *Java ME 8 SDK*: tools e emulatore per lo sviluppo delle applicazioni embedded.

- *Plugins*: plug in aggiuntivi per gli ambienti di sviluppo (Netbeans e Eclipse sono quelli più completi)

. Inoltre sono messi a disposizione degli emulatori di dispositivi per testare le proprie applicazioni senza bisogno delle piattaforme fisiche. L'emulatore può essere fatto partire dall'ambiente di sviluppo o dalla linea di comando. Quando viene fatto partire, viene visualizzata una schermata con tantissime informazioni relative al dispositivo emulato. E' possibile anche simulare eventi per il dispositivo grazie all'External Events Generator, in modo da fare i test che servono.

Figura 4.4: Tipologie di connessione supportate

4.3 Architettura delle applicazioni

Una applicazione è una entità che viene fatta partire dall'Application Management Software (AMS). L'applicazione deve avere una classe che estende la classe *javax.microedition.midlet.MIDlet*. Quando una applicazione viene invocata, ha bisogno ovviamente della Java Virtual Machine per funzionare.

4.3.1 Ciclo di vita

Come detto, l'applicazione deve estendere la classe *MIDlet*. Il ciclo di vita di una generica applicazione può essere descritto da una macchina a stati come in figura 4.5, i cui stati sono:

- *Active*: l'applicazione funziona normalmente. Si entra in questo stato subito prima dell'esecuzione del metodo *MIDlet.startApp()* da parte dell'AMS.

- *Paused*: l'applicazione è inizializzata e in stato di inattività. Non deve utilizzare quindi nessuna risorsa condivisa. Si entra in questo stato

quando l'applicazione viene creata con il comando *new*. Il costruttore della classe non ha argomenti e ritorna subito. Se viene lanciata un'eccezione, si passa subito allo stato destroyed.

- *Destroyed*: l'applicazione ha rilasciato tutte le sue risorse ed è terminata. Si entra in questo stato solo una volta, e si può fare in due modi. Il primo è quando l'AMS chiama il metodo *MIDlet.destroyApp()* e ritorna, rilasciando tutte le risorse. Il secondo modo è quando il metodo *MIDlet.notifyDestroyed()* ritorna con successo. L'applicazione deve aver svolto le stesse mansioni che svolgerebbe *MIDlet.destroyApp()* prima di chiamare questo metodo.

Una tipica sequenza di esempio per mostrare i vari passi fatti durante l'esecuzione di una applicazione sono i seguenti:

- L'AMS crea una nuova istanza dell'applicazione. Viene chiamato il costruttore di default senza argomenti. Ci si trova nello stato paused.
- L'AMS decide che l'applicazione può partire, quindi si passa nello stato active e viene chiamato il metodo *MIDlet.startApp()*. L'applicazione acquisisce le risorse che le servono e comincia i suoi compiti.
- L'AMS decide che l'applicazione deve essere terminata (per vari motivi: mancanza di risorse per applicazioni con priorità più alta, chiamata del metodo *MIDlet.notifyDestroyed()*..) e chiama il metodo *MIDlet.destroyApp()*. L'applicazione, se è stata progettata in modo corretto, rilascia tutte le risorse che aveva acquisito.

4.4 Alcuni esempi

Per poter sviluppare applicazioni embedded, è necessario installare l'SDK di Java SE, e successivamente Java ME sul computer. Successivamente va installato l'ambiente di sviluppo (ad esempio NetBeans) e i relativi plug ins. Di seguito viene mostrati semplici esempi applicativi che hanno lo scopo di mostrare alcune API e modelli di programmazione di Java Embedded. E' stato usato come ambiente di sviluppo NetBeans 8.0 e sono stati usati gli emulatori messi a disposizione e un Raspberry Pi.

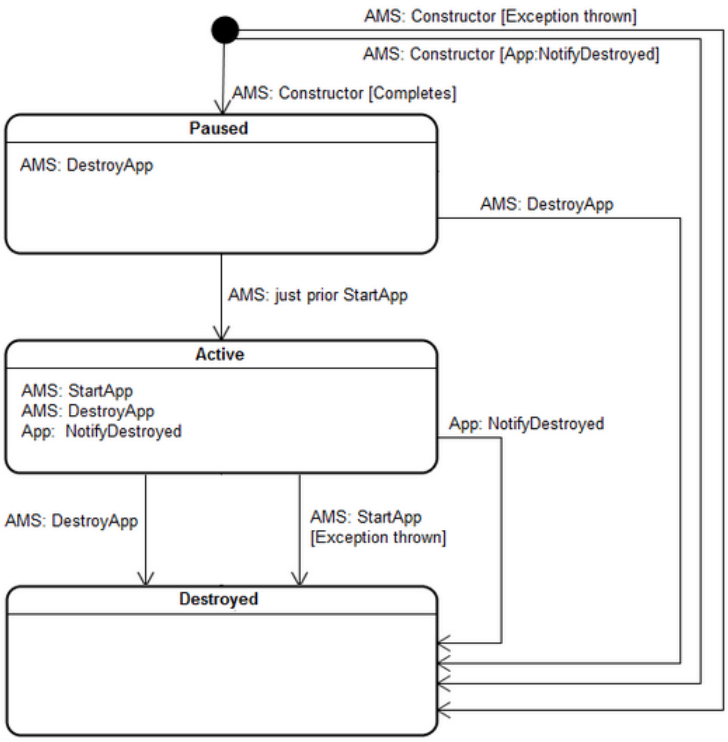

Figura 4.5: Ciclo di vita delle applicazioni

4.4.1 GPIO

Ora verrà mostrato come gestire i pin GPIO. Lo scenario è il seguente: vogliamo accedere in lettura/scrittura a dei dispositivi connessi ai pin GPIO di una board Raspberry-Pi o Arduino (o altro). Non c'è un solo modo per farlo, ma quello più comodo risulta utilizzare il DeviceManager e il late binding. Questo permette di gestire i pin allo stesso modo, a prescindere dal fatto ad esempio che sia di input o di output. In questo modo infatti non viene stabilita la tipologia del pin a tempo di compilazione, ma a tempo di esecuzione. Ovviamente per evitare errori a tempo di esecuzione bisogna usare il pin nella maniera appropriata. Supponiamo ad esempio di voler accendere un led. E' sufficiente collegarlo a un pin e settarne il valore logico alto. Per prima cosa è necessario dichiarare l'ID del pin di interesse, e nel caso del Raspberry pi, possiamo scrivere:

```
private static final int LED_ID = 23;
```

Per poter settare il valore alto, è prima necessario aprire il pin, e per farlo si scrive:

```
GPIOPin led = (GPIOPin) DeviceManager.open(LED_ID);
```

Ora è possibile settare il valore alto o basso. Per accendere il led serve il valore alto

```
led.setValue(true);
```

In questo modo in maniera molto semplice è possibile settare un valore alto o basso a un pin GPIO in modo da interagire con il componente fisico che è collegato.

4.4.2 Eventi

Una funzione che può avere risvolti molto utili è la possibilità di rilevare eventi. Continuando l'esempio dei pin, è possibile monitorare lo stato dei pin di input (ad esempio possiamo pensare di collegarci un bottone), e di svolgere determinate azioni. Ad esempio vogliamo eseguire qualcosa quando viene premuto un bottone. Una classe implementa l'interfaccia *PinListener*, e si occupa di gestire i pin e di gestire gli eventi. Per prima cosa va inizializzato il pin come nell'esempio precedente, e in più va settato il listener, scrivendo:

```
button.setInputListener(this);
```

In questo modo si verrà notificati nel caso di eventi. Se ad esempio si vuole fare una semplice stampa di quando il bottone viene premuto, è necessario implementare il metodo:

```
void valueChanged(PinEvent event)
```

Ora verrà mostrato il codice per realizzare una stampa ogni qual volta il pin cambia il suo stato:

```
public void valueChanged(PinEvent event) {
        GPIOPin pin=(GPIOPin) event.getDevice();
        try {
            if (pin == button) {
                System.out.println("Pin status changed");
            }
        } catch (IOException ex) {
            System.out.println("IOException: " + ex);
        }
    }
```

4.4.3 Connessione

Ora si vuole fare in modo che due dispositivi si connettano per scambiarsi informazioni. In questo esempio verranno usate le socket, ma sono disponibili diversi metodi di connessione, come mostrato precedentemente in figura 4.4. Ci sarà un server che accetterà connessioni da parte del client e verranno scambiati dei messaggi.

- *Server*: il MIDlet si occupa solamente di creare il server e di farlo partire. Quando lo crea gli passa la porta e sè stesso. La porta perchè è lì che il server dovrà aspettare la connessione, sè stesso perchè in questo modo il server potrà comunicare al MIDlet quando verrà distrutto in modo da terminare così l'applicazione. Il server implementa Runnable in quanto avrà un thread. Nel metodo run() del thread il server svolgerà i suoi compiti. Per prima cosa va creata e aperta la connessione:

```
System.out.println ("Waiting for connection on port "
+ portString);
ServerSocketConnection scn =
 (ServerSocketConnection) Connector.open
  ("socket://:"+ portString)
```

Poi il server deve mettersi in attesa di una connessione:

```
 SocketConnection sc =
  (SocketConnection) scn.acceptAndOpen ();
 System.out.println ("Connection accepted");
```

Dopo aver accettato una richiesta, deve prepararsi a mandare un messaggio il client, e a riceverne uno. Per farlo, deve inizializzare lo stream per l'input e per l'output.

```
  InputStream is = sc.openInputStream ();
  OutputStream os = sc.openOutputStream ();
```

Per quanto riguarda l'invio del messaggio, si può pensare di creare una oggetto Sender (con un thread) a cui viene passato un messaggio da inviare e l'output stream del server su cui scrivere. Per quanto riguarda la ricezione, questa può essere effettuata sempre nel metodo run del thread del server, leggendo dall'inputstream (su cui scriverà il client).

- *Client*: il client non differisce di molto dal server. Innanzitutto deve conoscere l'indirizzo del server e la porta su cui sta aspettando la connessione. Il client viene creato nel MIDlet come nel caso precedente, e si comporta come il server (e usare l'API corretta, che però non cambia di molto la sua struttura). L'unica differenza è che ovviamente non deve attendere richieste di connessioni, ma deve solo connettersi specificando indirizzo del server e porta:

```
SocketConnection sc =
 (SocketConnection) Connector.open
  ("socket://"+ address +":" + portString);
```

4.4.4 I2C

Java Embedded offre molte API per gestire le periferiche. Molti sensori e altri dispositivi di rilievo per IoT possono essere connessi alle board tramite il protocollo I2C. Per farlo, verrà usato nuovamente il DeviceManager, semplificando la programmazione. Rispetto all'esempio del led, verrà anche creato un file di configurazione contenenti le informazioni relative al dispositivo connesso (che cambiano da dispositivo a dispositivo, e vanno cercate nel relativo datasheet). Va quindi creato il file di configurazione:

```
I2CDeviceConfig config = new I2CDeviceConfig
(i2cBus, address, addressSizeBits, serialClock);
```

Dove i2cBus, address, addressSizeBits e serialClock sono tutti interi, e sono relativi alla board e al dispositivo connesso. i2cBus rappresenta il bus I2C a cui il dispositivo è connesso, address è l'indirizzo del dispositivo connesso, addressSizeBits è la dimensione in bit dell'indirizzo, serialClock è la frequenza di funzionamento relativa al dispositivo connesso con I2C. Successivamente, come nel caso del led, va usato il metodo open, passandogli però il file di configurazione:

```
I2CDevice device = (I2CDevice)DeviceManager.open(config);
```

In questo modo è possibile accedere a un dispositivo connesso con I2C. Nel caso di dover gestire altri protocolli, il processo è molto simile, poichè vanno solo cambiate le informazioni passate nel file di configurazione. Dato che protocolli diversi sono gestiti in modo simile, viene semplificata di molto la programmazione poichè non bisogna preoccuparsi di aspetti di basso livello.

Capitolo 5

Conclusioni

Lo scopo dell'elaborato era di introdurre la visione di Internet of Things e valorizzare l'importanza dello sviluppo di middleware per supportarla. Dopo un'introduzione relativa agli ambiti applicativi e alle problematiche da affrontare in questa visione, sono state introdotte anche le relative tecnologie abilitanti. Successivamente sono stati analizzati gli aspetti importanti dei middleware e sono stati presentati alcuni esempi. Infine si è approfondita la piattaforma Java Embedded. Questa piattaforma si è dimostrata particolarmente adatta a questa visione, poichè permette di risolvere alcuni dei principali problemi legati allo sviluppo, in particolare all'eterogeneità dell'hardware relativa ai dispositivi in gioco. L'approccio con macchina vituale risulta molto efficace poichè permette di lavorare con dispositivi con hardware diverso con lo stesso linguaggio di programmazione, limitando l'impatto che tutto questo ha sul programmatore. Lo sviluppo di piattaforme di questo tipo in questi anni è molto frequente, e questo sottolinea l'importanza che questa nuova visione avrà nel futuro e l'impatto che avrà sulla vita di tutti i giorni.

Ringraziamenti

Al termine di questo percorso, è doveroso ringraziare chi mi ha sempre supportato e sostenuto. In particolare la mia famiglia, che mi fornito i mezzi economici per arrivare fino a questo punto ma che soprattutto ha creduto in me dal primo all'ultimo momento senza mai smettere di farlo, sopportandomi nei momenti più duri. La mia ragazza, che mi è stata sempre accanto con amore, anche lei sopportandomi sempre nei momenti più difficili e condividendo la sua vita con la mia. Infine i miei amici, che mi considerano ancora nonostante i miei tantissimi "non posso fare tardi che domattina devo studiare", e ai miei amici e compagni di università che hanno rese più corte le ore di lezione e i viaggi in treno.

Bibliografia

[1] mbed. https://mbed.org/.

[2] Oracle java me embedded. http://www.oracle.com/technetwork/java/embedded/javame/embed-me/overview/javame-embedded-overview-2148916.html.

[3] What is ibeacon? a guide to beacons. http://www.ibeacon.com/what-is-ibeacon-a-guide-to-beacons/.

[4] Introduzione agli rfid, 2011. http://www.rfid.fub.it/edizione_2/Parte_I.pdf.

[5] Wireless sensor network, 2011. http://wsnsystem.blogspot.it/2011/12/wsn-environment.html.

[6] A brief history and future of the internet of things, 2014. http://www.startupsmart.com.au/technology/a-brief-history-and-future-of-the-internet-of-things/2014050912252.html.

[7] J. Al-Jaroodi, J. Aziz, and N. Mohamed. Middleware for rfid systems: An overview. In *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, volume 2, pages 154–159. IEEE, 2009.

[8] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[9] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta. Role of middleware for internet of things: A study. *International Journal of Computer Science and Engineering Survey*, 2(3), 2011.

[10] L. di Indicod-Ecr. Parte 2: La tecnologia rfid in standard epc, 2011. `http://indicod-ecr.it/documenti/epc/report/report_new_parte_(2).pdf`.

[11] A. Elahi and A. Gschwender. *ZigBee Wireless Sensor and Control Network*. Pearson Education, 2009.

[12] D. Evans. The internet of things. *How the next evolution of the internet is changing everything, White paper. CISCO IBSG*, 2011.

[13] O. Garcia-Morchon, S. Kumar, R. Struik, S. Keoh, and R. Hummen. Security considerations in the ip-based internet of things. 2013.

[14] A. Gupta and J. Singh. A study on clustering architecture and protocols in wireless sensor network. 2014.

[15] A. Koubaa, A. Cunha, and M. Alves. A time division beacon scheduling mechanism for ieee 802.15. 4/zigbee cluster-tree wireless sensor networks. In *Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on*, pages 125–135. IEEE, 2007.

[16] J.-S. Lee, Y.-W. Su, and C.-C. Shen. A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*. IEEE, 2007.

[17] X. Li and S. Moh. Middleware systems for wireless sensor networks: A comparative survey. 2014.

[18] G. Madlmayr, J. Langer, C. Kantner, and J. Scharinger. Nfc devices: Security and privacy. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 642–647. IEEE, 2008.

[19] F. Mattern and C. Floerkemeier. From the internet of computers to the internet of things. In *From active data management to event-based systems and more*, pages 242–259. Springer, 2010.

[20] S. McHugh and K. Yarmey. *Near Field Communication: Recent Developments and Library Implications*. Morgan & Claypool Publishers, 2014.

[21] M. Moloney. State of the art for near field communication: Security and privacy within the field. *Moloney, M.(2013)"State of the Art in Near Field Communication." in Security, Privacy, Trust, and Resource Management in Mobile and Wireless Communications, edited by Danda B. Rawat, Bhed B. Bista and Gongjun Yan*, 2013.

[22] H. Ning and Z. Wang. Future internet of things architecture: like mankind neural system or social organization framework? *Communications Letters, IEEE*, 15(4):461–463, 2011.

[23] T. Obaid, H. Rashed, A. Abou-Elnour, M. Rehan, M. M. Saleh, and M. Tarique. Zigbee technology and its application in wireless home automation systems: A survey. *International Journal of Computer Networks & Communications*, 6(4), 2014.

[24] J. Radhika and S. Malarvizhi. Middleware approaches for wireless sensor networks: An overview. *IJCSI International Journal of Computer Science Issues*, 9(6), 2012.

[25] A. Sheoran and S. Singh. Overview of near field communication (nfc). 2014.

[26] D. Singh, G. Tripathi, and A. J. Jara. A survey of internet-of-things: Future vision, architecture, challenges and services. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 287–292. IEEE, 2014.

[27] P. N. Tran and N. Boukhatem. Ip-based rfid architecture and location management. In *Software, Telecommunications and Computer Networks, 2008. SoftCOM 2008. 16th International Conference on*, pages 95–99. IEEE, 2008.

[28] B. Venners. The java virtual machine. `http://www.artima.com/insidejvm/ed2/jvm2.html`.

[29] S. Vongsingthong and S. Smanchat. Internet of things: A review of applications and technologies.

[30] R. H. Weber. Internet of things–new security and privacy challenges. *Computer Law & Security Review*, 26(1):23–30, 2010.

[31] L. Xu, W. He, and S. Li. Internet of things in industries: A survey. 2014.

[32] L. Yan, Y. Zhang, L. Yang, and H. Ning. *The Internet of Things: From RFID to the Next-Generation Pervasive Networked Systems*. Taylor & Francis, 2008.

[33] P. Yang, W. Wu, M. Moniri, and C. C. Chibelushi. Efficient object localization using sparsely distributed passive rfid tags. *Industrial Electronics, IEEE Transactions on*, 60(12):5914–5924, 2013.

[34] A. Zanella, N. Bui, A. P. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 2014.

[35] A. Zanella and M. Zorzi. Reti di sensori: dalla teoria alla pratica. *SIGNET (Special Interest Group on NETworking) presso il Dipartimento di Ingegneria dell'Informazione dell'Università di Padova*, 2006.