Università degli studi
di Modena e Reggio Emilia

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Progetto e Sviluppo di un'Applicazione per il Monitoraggio e l'Archiviazione di Dati di una Wireless Sensor Network

Andrea Izzo

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Correlatore:
Dott. Fabio Bertarelli

Anno Accademico 2008/2009

Ringraziamenti

Ringrazio la mia famiglia per il supporto e sostegno datomi in tutti questi anni. Ringrazio anche il Prof. Riccardo Martoglia e il Dott. Fabio Bertarelli per la loro continua disponibilità e assistenza.

Parole chiave

Wireless Sensor Network
TinyDB
SQLite
SimpliciTI
Php

Indice

Introduzione **1**

I Il caso di studio **5**

1 Wireless Sensor Network **7**
1.1 Le Wireless Sensor Network 7
1.1.1 Architettura di un nodo sensore 9
1.1.2 Protocolli di rete 11
1.2 Campi di applicazione e vincoli di una WSN 13

2 TinyOS **16**
2.1 Studio del sistema operativo TinyOS 16
2.1.1 NesC 17
2.2 TinyOS 1.x 18
2.3 TinyOS 2.x 20
2.3.1 Hardware Abstraction Architecture (HAA) 21
2.4 TinyDB 24
2.4.1 TinySQL 25
2.5 Considerazioni su TinyDB e TinyOS 27

3 SQLite e DB studiati **28**
3.1 SQLite 28
3.2 MySQL 31
3.3 PostgreSQL 33
3.4 Motivazioni e considerazioni sulla scelta di SQLite 34

4 Tecnologie usate **36**
4.1 Texas Instruments EZ430-RF2500 36
4.1.1 Il microcontrollore MSP430 38
4.1.2 Il transceiver CC2500 39
4.2 Protocollo SimpliciTI 40
4.2.1 Architettura del protocollo 40

4.2.2 Dispositivi logici 42
4.2.3 Frame di trasmissione 43
4.3 PHP 44
4.3.1 PHP Data Objects (PDO) 44

II Progetto e implementazione dell'applicazione 45

5 Progetto 46
5.1 Requisiti funzionali 46
5.2 Use Case Diagram 49
5.3 Scenario: monitoring sensori o visualizzazione dell'archivio 50
5.3.1 Caso d'uso in forma tabellare 51
5.3.2 Activity Diagram 53
5.4 Struttura dei database 54
5.4.1 Schemi logici 55

6 Implementazione 57
6.1 Visione generale 57
6.2 Software Wrapper 58
6.2.1 Funzionamento del codice 59
6.3 Software AP 61
6.4 Software ED 63
6.5 Interfaccia Web 65
6.5.1 SensiNet.php 66
6.5.2 SensiNet_rep.php 66
6.5.3 login.php 67
6.5.4 sensori.php 68
6.5.5 home.php 68
6.5.6 cerca.php 70
6.5.7 Admin.php 70
6.5.8 inserimento.php 71
6.6 Prove sperimentali 72

7 Conclusioni e sviluppi futuri 74

III Appendici 77

8 Codice 78
8.1 Software wrapperdb.sqlite 78
8.2 Software AP e ED 87
8.2.1 Codice AP modificato 87
8.2.2 Codice ED standard 101

8.3 Applicazione Web 109
8.3.1 SensiNet.php 109
8.3.2 SensiNet_rep.php 111
8.3.3 login.php 113
8.3.4 sensori.php 117
8.3.5 home.php 121
8.3.6 cerca.php 131
8.3.7 home.php 135
8.3.8 inserimento.php 144

Elenco delle figure

1.1 Esempio tipico di WSN 8
1.2 Schema a blocchi di un nodo 9
1.3 Esempi di rete gerarchica e flat 12
1.4 Esempio di rete mesh 13
1.5 Esempio di monitoraggio della salute di una barriera corallina . . 14

2.1 Struttura del Hardware Abstraction Architecture 22

3.1 Struttura di SQLite 31

4.1 EZ430-RF2500 37
4.2 Architettura dei livelli del protocollo SimpliciTI 40
4.3 Esempio di rete con protocollo SimpliciTI 42
4.4 Frame del protocollo SimpliciTI 43

5.1 Caso d'uso di un utente nel dettaglio 50
5.2 Activity Diagram per il monitoring sensori e visualizzazione archivio 53
5.3 Organizzazione e struttura dei database 54

6.1 Sistema complessivo 58
6.2 Diagramma di flusso del software AP 61
6.3 Diagramma di flusso del software ED 64
6.4 Esempio della pagina SensiNet.php 66
6.5 Esempio della pagina SensiNet_rep.php 67
6.6 Esempio della pagina login.php 68
6.7 Esempio di visualizzazione delle tesi nella pagina home.php 69
6.8 Esempio della tabella inserimento utente della pagina Admin.php 71
6.9 Tabella delle performance tra i vari DBMS 72

Introduzione

I recenti progressi tecnologici nei sistemi micro-elettro-meccanici, nelle comunicazioni wireless e nell'elettronica digitale hanno permesso lo sviluppo di sensori a basso consumo energetico e ottimo livello di efficienza. L'integrazione di funzionalità elaborative, sensoristiche, attuative, comunicative e di memorizzazione all'interno di questi dispositivi miniaturizzati prende il nome di *Wireless Sensor Network (WSN)* (o anche reti di sensori wireless).

Una Wireless Sensor Network è di fatto un insieme di piccoli dispositivi elettronici autonomi in grado di prelevare dati dall'ambiente circostante e di comunicare tra loro. Fino ad ora la possibilità di cooperare con altri sistemi era limitata dall'ausilio di cavi per la trasmissione delle informazioni, ma grazie alle caratteristiche principali delle reti di sensori wireless ora si ha la possibilità di costruire una rete nella quale i dispositivi possano comunicare tra loro via radio, in più la dimensione ridotta dei nodi sensore, i costi contenuti e la facile installazione dei dispositivi, la scarsa manutenzione richiesta e la possibilità di posizionare dispositivi della rete anche su parti in movimento hanno portato velocemente al successo e all'espansione delle WSN.

Le WSN possono essere collocate nel contesto del *pervasive computing* (everywhere), dove l'elaborazione delle informazioni risulta integrata all'interno di oggetti e attività comuni, finora mai destinate a questo tipo di utilizzo. L'ambiente diviene così intelligente, capace di interagire anche in modo autonomo con l'uomo e con gli oggetti e parametri che lo compongono, permettendone un fine adattamento alle esigenze richieste. Le reti di sensori wireless hanno di fatti un campo applicativo veramente vasto che tocca i più numerosi settori industriali tra cui domotica, agricoltura, allevamento, logistica, controllo ambientale, edilizia, opere pubbliche, gestione e sorveglianza delle infrastrutture fino ad arrivare alla medicina e al mondo militare. Un esempio di utilizzo di una rete di

sensori può essere quello nel campo della tutela ambientale dove possono essere utilizzate per rivelare tempestivamente incendi boschivi, oppure per monitorare l'inquinamento di falde acquifere e laghi o, ancora, per controllare aree a rischio, come discariche, impianti industriali e chimici, centrali nucleari. Invece, un altro esempio è nel campo medico dove si possono sfruttare le WSN attraverso l'impianto nell'organismo di minuscoli nodi sensore in grado di monitorare i parametri vitali della persona e somministrare la terapia farmacologica in modo autonomo, quando necessario.

Le reti di sensori combinano le capacità di raccogliere informazioni dall'esterno, effettuare elaborazioni e comunicare attraverso un ricetrasmettitore, per realizzare una nuova forma di rete, che può essere installata all'interno di un ambiente fisico, sfruttando le dimensioni ridotte dei dispositivi che la compongono e il loro basso costo. Molti lavori di ricerca si sono inoltre focalizzati sulle modalità di interrogazione e aggregazione dei dati delle WSN. Risulta fondamentale disporre di strumenti in grado di permettere all'utente di richiedere in modo semplice le informazioni di cui necessita dalla rete di sensori.

L'obiettivo fondamentale di questa trattazione, dunque, sarà proprio quello di realizzare un'applicazione Web che implementi le modalità di monitorizzazione e archiviazione dei dati provenienti da una WSN rispettando le diverse caratteristiche che la compongono e mascherando all'utente la reale implementazione e le differenze architetturali.

La struttura di questa tesi è organizzata in tre macro-parti suddivise a loro volta in capitoli.
La prima parte analizza le tecniche necessarie alla comprensione del lavoro svolto ed è composta in quattro capitoli:

- il primo capitolo illustra una panoramica generale della struttura e delle caratteristiche principali delle Wireless Sensor Network;

- nel secondo capitolo si passa poi allo studio di *TinyOS*, un sistema operativo contenente il database *TinyDB*, sviluppato appositamente per le WSN, scelto originariamente come piattaforma per lo sviluppo della tesi, ma poi abbandonato per lasciar spazio al più diffuso *SQLite*;

- il terzo capitolo tratta, di fatto, dello studio dei vari DBMS presi in esame al momento della scelta del database e illustra le motivazioni della scelta di SQLite;

- il quarto capitolo chiude la prima parte e parla delle varie tecnologie software e hardware utilizzate in questa applicazione.

Nella seconda parte si entra nel dettaglio progettuale ed implementativo e si divide nei successivi tre capitoli:

- il quinto capitolo tratta della fase di sviluppo e progettazione dell'applicazione;

- il sesto capitolo illustra nel dettaglio l'implementazione spiegando il funzionamento del codice nel dettaglio delle varie parti che lo compongono.

- nel settimo capitolo vengono analizzate le prestazioni di SQlite per quanto riguarda il tempo di esecuzione e spazio occupato sul disco fisico.

La terza ed ultima parte riporta le appendici contentente tutti codici riportati per intero.

Parte I

Il caso di studio

Capitolo 1

Wireless Sensor Network

1.1 Le Wireless Sensor Network

Una Wireless Sensor Network (WSN)[1] è una rete di piccoli nodi (o motes) capaci di ospitare sensori con capacità di comunicazione wireless in grado di prelevare dati dall'ambiente circostante e di eseguire delle elaborazioni sullo stesso nodo grazie ad un processore on-board. Questi piccoli dispositivi sono prodotti e distribuiti in massa, ed hanno un costo di produzione trascurabile. Ognuno dei sensori ha una riserva d'energia limitata e non rinnovabile e, una volta messo in opera, deve lavorare autonomamente. Per ottenere la maggior quantità possibile di dati occorre effettuare una massiccia distribuzione di sensori (migliaia o decine di migliaia).

Le reti di questo tipo si sono rapidamente diffuse in quanto offrono una serie di innegabili vantaggi: la mobilità, che permette ai terminali di potersi muovere, la flessibilità ed i bassi costi di realizzazione. Tuttavia anche le reti wireless devono affrontare alcune problematiche. Una di queste è indubbiamente la capacità del mezzo trasmissivo, l'etere, che è unico e condiviso da tutti i nodi connessi. L'esistenza di un unico canale limita necessariamente il numero massimo di utenti che possono usufruire del servizio contemporaneamente. Allo stesso modo, la presenza di più utenti determina una riduzione della velocità di trasmissione, in quanto la capacità del canale trasmissivo deve essere suddivisa tra tutti coloro che ne stanno facendo uso. Non da meno è il problema della sicurezza: in assenza di specifici controlli, risulta facile per un attaccante intercettare le informazioni che viaggiano nell'etere o riuscire ad accedere a servizi anche senza autorizza-

zione. Occorre inoltre considerare che la qualità della comunicazione può venir influenzata anche da fattori esterni, come interferenze elettromagnetiche e ostacoli in movimento. Infine il consumo energetico degli apparati di trasmissione radio è tipicamente più elevato di quelli per la comunicazione via cavo.

Su ogni dispositivo è alloggiata una parte di controllo, una di comunicazione e una di sensoristica che permettono la costruzione di reti di sensori, in grado di essere dislocate su aree abbastanza vaste e capaci di comunicare tra loro tramite protocolli di comunicazione sviluppati ad hoc. Le reti di sensori possono migliorare in modo significativo la qualità delle informazioni: ad esempio possono garantire una elevata fedeltà, possono fornire informazioni in tempo reale anche da ambienti ostili e possono ridurre i costi di trasmissione delle stesse informazioni.

Una wireless sensor network è solo una parte di un sistema più complesso, definito Sistema WSN; esso, infatti, inizia dalla WSN vera e propria ma prosegue con il canale di comunicazione tra la WSN e una database di raccolta dati, che può essere un server internet, e l'interfaccia tra il database e l'utente finale; schematicamente un Sistema WSN può essere rappresentato come in **figura 1.1**.

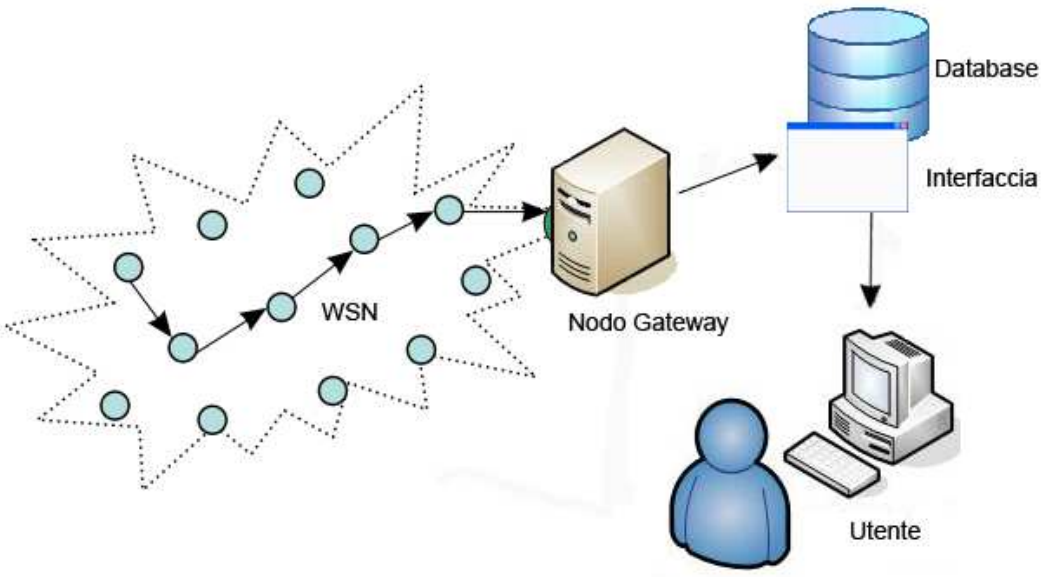


Figura 1.1: Esempio tipico di WSN

Quindi, ha senso parlare di WSN, e soprattutto, ha senso il suo utilizzo, se non rimane confinato alla semplice dimostrazione di funzionalità della stessa, ma è in grado tramite delle interfacce opportune di interagire con l'utente finale: solo in questo modo si può ritenere che il *sensing* dell'ambiente in questione sia ragionevolmente utile. Analizzando in dettaglio gli elementi che costituiscono una WSN, appare subito chiara la differenza tra i nodi della rete, preposti alla gestione dei sensori e al mantenimento della infrastruttura wireless della rete, e uno (o anche più) nodi, che hanno il compito di fare da collettori e trasmettere i dati ricevuti dagli altri nodi verso il server centrale; infatti ognuno di essi può interagire con gli altri nodi, secondo il protocollo di comunicazione adottato, in modalità *flat*, *gerarchica* o *mesh* (vedi **Capitolo 1.1.2 Protocolli di rete**).

L'obiettivo primario di ogni nodo è comunque quello di inviare i propri dati verso un centro di raccolta, all'interno della Wireless Sensor Network, che viene definito *Gateway*: quest'ultimo ha il compito di inviare tutti i dati pervenuti ad un sistema centrale, generalmente un server, che funge da database, tramite comunicazione *wired* (ethernet, usb, LAN, firewire, etc..), oppure *wireless* (es. GPRS, UMTS, HDSPA etc..). Nelle Wireless Sensor Network più evolute il flusso dati può non limitarsi a quello monodirezionale descritto (dai nodi al Gateway), ma può essere omnidirezionale, ovvero possono essere trasmessi comandi da nodo a nodo, o dal server centrale (quindi dall'utente) ai nodi.

1.1.1 Architettura di un nodo sensore

Un nodo sensore è composto da quattro unità base:

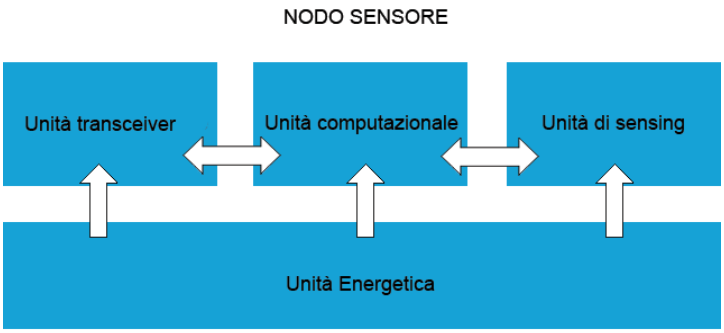


Figura 1.2: Schema a blocchi di un nodo

Unità di sensing
Un nodo di solito è composta da due sotto unità: sensore/attuatore e convertitore analogico-digitale (ADC); Un sensore, quindi, è in grado di rivelare e/o misurare grandezze ambientali trasformandole in un segnale elettrico opportuno, senza tuttavia poterle modificare. Un ruolo duale è svolto dagli attuatori, dispositivi in grado di agire sull'ambiente circostante in diversi modi (esempi di attuatori sono bracci meccanici, valvole, irrigatori, ma anche sirene d'allarme, altoparlanti, videoproiettori, caloriferi, ecc.). Un nodo sensore può comprendere un numero variabile di trasduttori e attuatori, anche di diversa natura. L'equipaggiamento del nodo sensore, quindi, ne determina le funzionalità, ma anche il costo, il consumo energetico e l'ingombro.

Il convertitore anologico-digitale è utilizzato invece per tradurre i segnali elettrici generati dai trasduttori in forma digitale in modo da consentirne l'elaborazione software. Analogamente, è spesso presente un'unità duale, ovvero un convertitore digitale-analogico (DAC), che trasforma segnali numerici generati dal microprocessore in segnali elettrici utilizzati per pilotare gli attuatori.

Unità computazionale
Al microprocessore è associata una piccola unità di immagazzinamento dati e gestisce le procedure che permettono la collaborazione del nodo con gli altri nodi della rete per portare a termine il task assegnato. Questo è possibile tramite l'integrazione di memoria RAM e ROM, utilizzate per ospitare il codice eseguibile del sistema operativo e dell'applicazione, nonchè i dati ed elaborati dall'applicazione; La ROM è con memoria di tipo Flash, cosicchè il sistema operativo e il software in esecuzione sul nodo siano programmabile e modificabile. La gestione e l'utilizzo delle memorie però porta ad alto consumo energetico, pertanto i blocchi di memoria integrati hanno capacità ridotte, limitate a poche decine di kbyte.

Unità transceiver
È l'unitàà che connette il nodo alla rete. Può essere un apparecchio ottico oppure un apparecchio a radio frequenza (RF). Solitamente tra tutti i componenti del nodo, il chip radio è il dispositivo che consuma la maggior parte dell'energia. Per ridurre il costo e il consumo energetico dei nodi, si utilizzano tipicamente

modulazioni ben consolidate e di bassa complessità, a discapito della capacità trasmissiva che è spesso limitata a qualche decina di kbit/s. Per limitare ulteriormente il costo finale del dispositivo, la modulazione radio avviene normalmente nelle bande comprese tra gli 868-870 MHz o nelle bande *ISM* (Industrial Scientific Medical) attorno ai 900 MHz e ai 2,4 GHz, per le quali non è richiesta licenza governativa.è possibile usare anche delle radio con basso duty-cycle anche se attualmente ci sono problemi nell'usare questi apparecchi in quanto viene consumata molta energia perle operazioni di spegnimento.

Unità energetica
È forse il componente più importante di un nodo sensore; a volte può essere supportata da una unità per il recupero di energia (ad esempio da fonte solare).

Software
Per quanto riguarda software la particole la struttura delle Wireless Sensor Network e le caratteristiche dei nodi richiedono lo sviluppo di applicazioni progettate appositamente per la piattaforma utilizzata. Questa argomentazione vale anche per il sistema operativo, a cui è richiesto ridottissima occupazione di memoria e basso consumo di energia durante l'esecuzione dei processi; altre prequisiti sono il consumo quasi nullo durante lo stato di inattività (*idle*) e gestione della concorrenza (accesso simultaneo di più thread alla stessa risorsa). Inoltre deve avere un supporto ai protocolli di rete (in senso di consumo energetico) ed infine l'accesso alle funzionalità di basso livello della piattaforma devono avvenire in maniera facile per mezzo di interfacce astratte. Come per il sistema operativo, il

software che definisce i servizi più avanzati della WSN deve occupare pochissimo spazio di memoria e deve risultare quanto più possibile efficiente in termini di consumi energetici. Questo comporta la necessità di progettare l'applicazione in modo da minimizzare quanto più possibile l'utilizzo delle diverse interfacce (radio, trasduttori, attuatori) e del processore, cosa che si traduce spesso in un *tradeoff* tra efficienza energetica del sistema e qualità del servizio offerto.

1.1.2 Protocolli di rete

Come spiegato in precendenza ogni nodo di una rete di sensori può interagire con gli altri secondo il protocollo di comunicazione adottato, in modalità *flat*,

gerarchica o *mesh*. I protocolli *flat* prevedono che tutti i nodi siano tra loro allo stesso livello di importanza, eccetto uno che funge da master o coordinatore (*sink*), che può o meno coordinare le trasmissioni degli altri, ma comunque ha il compito di trasferire i dati della WSN verso il server. La configurazione, che generalmente si viene a creare in questo tipo di rete, con pochi nodi, è detta a *stella*, perchè tutti i nodi comunicano col master in maniera diretta. I protocolli *gerarchici*, invece, prevedono l'attribuzione di livelli diversi tra i singoli nodi: in base alle metriche scelte, ogni nodo, che riceva un segnale di *beacon* (ovvero di saluto) da un altro, diventa lo slave e l'altro il master; a sua volta il nodo slave, generando un segnale di beacon, può diventare il master per un altro nodo. La configurazione che si viene a creare è detta ad albero.

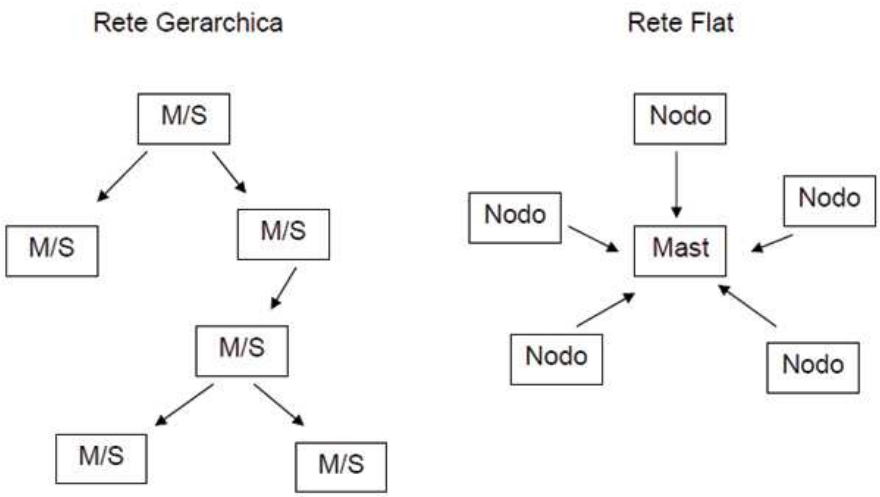


Figura 1.3: Esempi di rete gerarchica e flat

I protocolli per reti *mesh* prevedono che i nodi e i dispositivi costituenti la rete instaurano molteplici interconnessioni tra di loro. Ogni nodo è interconnesso con ogni altro elemento della rete e, in questo modo, vi è un incremento notevole dell'affidabilità della rete grazie alla molteplicità di percorsi disponibili per la trasmissione di un messaggio. È possibile, poi, grazie all'utilizzo di meccanismi di instradamento (*routing*), far determinare alla rete qual è il percorso più conveniente dal punto di vista energetico, quale è quello più breve e così via, in modo da gestire la rete in maniera molto più efficiente.

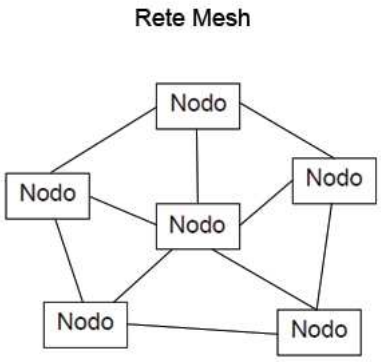


Figura 1.4: Esempio di rete mesh

1.2 Campi di applicazione e vincoli di una WSN

Sensori di diverso tipo all'interno di una WSN sono capaci di rilevare grandezze come temperatura, umidità, pressione, luce, movimento di veicoli, la composizione del terreno, livello di rumore e molte altre cose. Questo comporta ad un grandissimo numero di applicazioni possibili per le reti di sensori wireless nelle discipline scientifiche più diverse. Tali applicazioni possono essere raggruppate nelle seguenti categorie:

- **Applicazioni medico sanitarie**
  Gli utilizzi delle reti di sensori in questo campo sono rivolte a fornire un'interfaccia per le persone affette da handicap, al monitoraggio di dati fisiologici (riguardanti l'uomo), all'amministrazione ospedaliera sia essa relativa ai pazienti che ai medici (per una facile rintracciabilità). Inoltre è anche possibile usare i sensori per l'identificazione di allergie.

- **Sorveglianza militare**
  Le reti di sensori sono state utilizzate in principio per questo scopo. La facilità e rapidità di distribuzione, l'auto-organizzazione e la tolleranza ai guasti fanno di fatti una rete di sensori una tecnica promettente per applicazioni militari. Le possibili applicazioni vanno dal monitoraggio di forze alleate, equipaggiamenti e munizioni alla sorveglianza del campo di

battaglia. È possibile usare una rete di sensori in luoghi ostili per effettuare il riconoscimento e lo spostamento di nemici, la stima dei danni di una battaglia oppure il riconoscimento del tipo di attacco.

- **Monitoraggio ambientale**
  In questo ambito, le reti di sensori potrebbero essere usate per alcune applicazioni che riguardano, ad esempio, il monitoraggio del movimento di uccelli, piccoli animali, insetti e studiare i loro particolari habitat. È possibile anche effettuare il monitoraggio di una foresta e rilevare prontamente eventuali incendi o lo spostamento ed il dinamismo all'interno dei ghiacciai. Appartengono a questo settore anche lo studio di eventi naturali catastrofici quali la previsione e rilevazione di inondazioni, trombe d'aria, terremoti ed eruzioni vulcaniche. Le reti di sensori possono essere utilizzate anche nell'agricoltura di precisione. Tra gli obiettivi che è possibile raggiungere c'è quello di monitorare il livello dei pesticidi nell'acqua, il livello di erosione del terreno e il grado di inquinamento dell'aria.

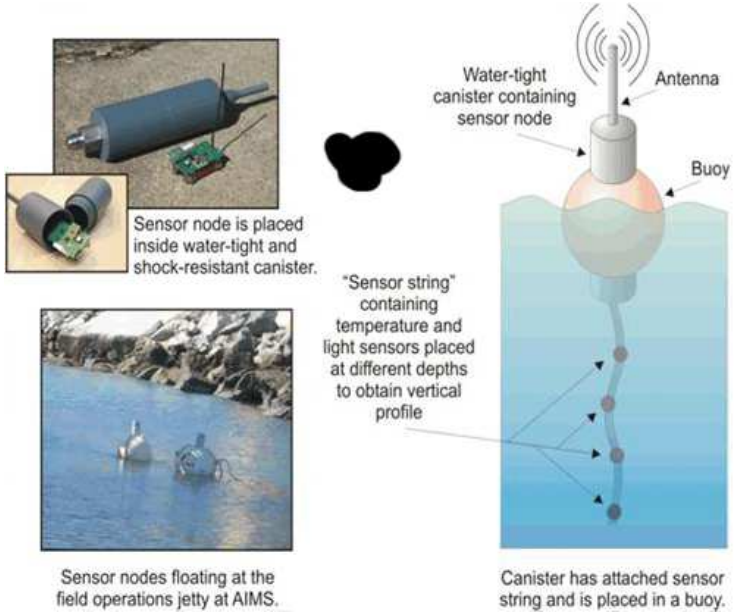


Figura 1.5: Esempio di monitoraggio della salute di una barriera corallina

- **Monitoraggio di apparecchiature industriali**
  I sensori wireless possono essere applicati a macchinari industriali per ana-

lizzarne il comportamento dei componenti sottoposti a stress meccanico, migliorarne le performance o prevenirne rotture e guasti.

- **Applicazioni commerciali**
  In questo ambito, i sensori potrebbero essere usati per le seguenti applicazioni: rilevamento del furto di auto: nodi sensore sono stati disposti per scoprire e identificare minacce che avvengono in una determinata zona geografica. rilevamento della posizione e del movimento di veicoli: tramite una rete di sensori è possibile monitorare la posizione di un'auto in movimento. È inoltre possibile monitorare il traffico su strade particolarmente transitate; controllo dell'ambiente in ufficio: di solito il sistema di gestione del riscaldamento o dell'aria condizionata è centralizzato, quindi con una rete di sensori nelle stanze è possibile controllare la temperatura e la distribuzione dell'aria calda nelle varie zone.

Le capacità e le qualità delle trasmissioni tra i sensori di una rete wireless sono però fortemente vincolate alle condizioni degli ambienti in cui sono costruite. In particolare i fattori che condizionano in modo significativo la qualità delle trasmissioni possono essere la distanza tra i vari nodi ed eventuali ostacoli presenti tra essi; potenza di trasmissione e possibili disturbi elettromagnetici e problemi legati all'hardware come la compatibilità tra varie architetture dei nodi o problemi dovuti all'alimentazione.

Il transceiver dei sensori in questione possono fornire alcuni parametri quali ad esempio l'LQI (*Link Quality Indicator*) e l'RSSI (*Received Signal Strength Indicator*) che acquistano una notevole importanza qualora vengano utilizzati come indici di grandezze in algoritmi complessi per WSN. L'RSSI ad esempio varia in funzione della distanza e quindi può diventare un ottimo alleato nei campi del tracking e della localizzazione, dove sono proprio le distanze, fisse o variabili, fra i nodi a farla da padrone (con le dovute calibrazioni per avvicinarsi sempre più ad un valore esatto privo di disturbi).

Capitolo 2

TinyOS

2.1 Studio del sistema operativo TinyOS

TinyOS[2] è un sistema operativo opensource realtime basato su moduli, sviluppato per gestire reti wireless di sensori. È sviluppato dall'Università della California, Berkley, in collaborazione con il centro ricerche Intel. L'attuale release ufficiale di TinyOS è la 2.1.0, rilasciata il 22 Agosto 2008. La prima release di TinyOS risale al 2002; è stato sviluppato originariamente per i *Mica Motes* di Berkeley, ma attualmente supporta più di dieci diverse piattaforme e numerosi tipi di *sensor board*.

TinyOS cerca di porre rimedio ai limiti e ai vincoli stringenti delle WSN; per questo motivo, è stato concepito in modo da introdurre caratteristiche di efficiente gestione energetica, basso carico computazionale, dimensioni ridotte, modularità e supporto alla concorrenza. È completamente progettato in linguaggio *NesC*, una variante del linguaggio di programmazione C basata su eventi (eventdriven), infatti l'applicazione è composta da diversi componenti la cui elaborazione viene avviata al verificarsi di un particolare evento. Questo approccio si contrappone al tradizionale paradigma basato su stack e switching del contesto di esecuzione: quando il sistema è in stato di *idle* non esegue alcuna operazione, consumando minime quantità di energia, in questo modo pur permettendo la concorrenza, evita l'attività di *contextswitching* tipica dell'approccio tradizionale.

Una tipica applicazione TinyOS based ha le dimensioni di qualche decina di KB, dei quali soltanto 400 byte riguardano il sistema operativo: è proprio dalla

dimensione decisamente ridotte che deriva il nome TinyOS. Un'altra importante caratteristica del TinyOS è l'assenza del kernel (si configura come un framework avanzato per applicazioni embedded). Per questo motivo esso si differenzia in modo consistente dai sistemi operativi generalpurpose, integrandosi direttamente con le applicazioni sviluppate; è quindi compito di ciascun componente controllare direttamente le risorse che utilizza, prevedendo autonomamente i meccanismi necessari a risolvere conflitti dovuti a impiego contemporaneo della periferica da parte di moduli diversi. Tuttavia, questi anni di esperienza hanno portato i ricercatori di Berkeley a reingegnerizzare e riprogettare alcuni aspetti basilari del sistema operativo creando la versione TinyOs 2.x, le due versioni però non risultano compatibili a livello di software eseguibile, pertanto programmi scritti per la versione 1.x non funzionano sulla nuova versione, esattamente come per la situazione inversa.

2.1.1 NesC

NesC (*Network embedded system*) è una variante del linguaggio C open-source sviluppato per la programmazione dei nodi di una WSN. È stato sviluppato dall'università di Berkeley sotto i termini della *Berkeley Software Distribution License* ed è stato utilizzato per la scrittura del sistema operativo TinyOS.

Per alcuni versi questo linguaggio è a tutti gli effetti un'estensione del C con l'aggiunta dell'implementazione degli eventdriven: *sleep* per la maggior parte del tempo e *awake* solo durante la fase di elaborazione, in accordo con la politica di utilizzo delle risorse di alimentazione del sensore.

Nel momento in cui un'applicazione viene compilata, i componenti di TinyOS vengono compilati insieme ad essa e il risultato costituisce l'intero software del sensore rendendo impossibile installare più applicazioni indipendenti sullo stesso sensore. Questo approccio consente un ingente risparmio di energia e di memoria, tuttavia limita la versalità, ad esempio limitando le operazioni riguardanti i puntatori, infatti non è possibile effettuare linking dinamico di librerie esterne o riconfigurazione dinamica di parte del codice presente sul sensore.

Un'altro dei concetti principali che sta dietro a NesC è la separazione dell'implementazione dalla composizione delle componenti, di fatto un'applicazione NesC è un insieme di componenti collegati tramite interfacce (tra i componenti

che compongono un'applicazione ci sono anche quelli del sistema operativo stesso) le quali sono bidirezionale e possono essere fornite o usate dalle componenti. Le interfacce fornite rappresentano le funzionalità che una componente fornisce agli utenti (i comandi); le interfacce usate rappresentano le funzionalità che una componente necessita per effettuare il proprio lavoro (gli eventi) . Questo meccanismo è ridondante, tuttavia raggruppare comandi ed eventi relativi a un servizio in interfacce rende il codice più leggibile. Comandi ed eventi al momento del linking statico vengono tradotti in chiamate a funzioni, quindi semanticamente la differenza è minima.

Esistono due tipi di implementazione di componenti: i moduli e le configurazioni.

I moduli implementano le funzionalità delle interfacce del modulo attraverso codice C con alcune estensioni. Il modulo implementa inoltre gli eventi e i comandi in modo molto simile a come vengono implementati i sottoprogrammi in C, ad esclusione del fatto che, prima della definizione dell'evento o del comando, bisogna inserire il nome dell'interfaccia relativa, e per alcune macro utilizzate come valori di ritorno. All'interno di un'implementazione è possibile sollevare eventi (tipicamente verso moduli di livello più alto) tramite la parola chiave *signal*, mentre è possibile invocare comandi (tipicamente su componenti più vicini all'hardware) tramite la parola chiave *call*. Ogni modulo contiene il suo stato sotto forma di variabili locali dichiarate alla stessa maniera delle variabili in C.

Le configurazioni implementano componenti dichiarando una serie di sottocomponenti e definendo i collegamenti tra le interfacce di questi sottocomponenti. Ogni applicazione è composta da una configurazione globale che definisce la connessione dei vari componenti.

2.2 TinyOS 1.x

La componente principale presente in ogni applicazione TinyOS 1.x è lo *Scheduler* il quale è incaricato di mandare in esecuzione i task dei vari componenti secondo una strategia *FIFO* (First In First Out). La schedulazione ammette due livelli di priorità: quello più basso è assegnato ai task che non si possono interrompere fra loro, mentre quello più alto è dedicato agli eventi che possono interrompere i task.

Il modello a componenti di TinyOS è dotato di *command handlers* e *event handlers*. I componenti comunicano tra loro invocando comandi (gestiti dal *command handlers*) e sollevando eventi (gestiti dall'*event handlers*). Comandi ed eventi vengono eseguiti al livello alto di priorità dello scheduler. A questi si aggiunge una serie di task, che vengono eseguiti a livello basso di priorità. Ogni componente inoltre contiene un *frame*. Questa è l'area dati del componente e viene allocata staticamente. Tipicamente gli eventi vengono sollevati da componenti più vicini all'hardware verso componenti meno vicini, mentre i comandi vengono invocati in verso opposto.

I componenti possono essere suddivisi in tre categorie:

- *Hardware abstractions*, questi componenti mappano le funzionalità fornite via software sulle funzionalità fornite dall'hardware creando un'astrazione dello stesso utilizzabile dai moduli superiori;
- *Synthetic hardware*, questi moduli simulano il comportamento di hardware più sofisticato di quello realmente presente sul sensore;
- *High level software component*, questi componenti sono quelli di livello più alto e si occupano di eseguire algoritmi che prescindono dal particolare hardware. Ad esempio un componente che legge/scrive un bit su un canale radio è un'astrazione hardware, mentre un componente che compone 8 bit e li invia a quelli di livello superiore è un hardware sintetico, mentre il componente che implementa il protocollo di routing è un componente di alto livello.

Split-phase operations

In TinyOS ogni operazione non banale è una *split-phase operation*, ovvero la chiamata e l'operazione di ritorno sono in realtà due funzioni separate. Una funzione viene invocata chiamando un comando su un componente, questo avvia un task e ritorna immediatamente. Alla terminazione del task viene sollevato un evento sul componente che ha invocato l'operazione. Un tipico esempio è l'invio di un pacchetto sulla rete. Ogni componente implementa una delle due metà delle operazioni split-phase previste (o il comando o l'evento).

Active Messages

La tecnologia di rete utilizzata da TinyOS prende il nome di *Active Messages*, si tratta di un protocollo di comunicazione orientato al datagramma, non affidabile e single-hop. Questo sistema permette di inviare messaggi a tutti (broadcast) o a un singolo nodo tra i nodi vicini (protocolli di routing sono implementati ai livelli superiori, quindi ha solo un meccanismo di indirizzamento verso i nodi vicini). Active messages una tecnologia molto leggera, infatti non specifica meccanismi connection oriented, ogni pacchetto è un'entità indipendente. Esso contiene l'identificatore di un handler da richiamare sulla macchina di destinazione e il payload del pacchetto. Questo sistema permette di evitare il complesso utilizzo di buffer delle implementazioni del protocollo TCP/IP. Il principale svantaggio dell'approccio è che tutti i nodi comunicanti devono avere lo stesso software o quantomeno implementare un componente che definisca lo stesso handler.

2.3 TinyOS 2.x

TinyOS 2.x si presenta come una riscrittura completa del TinyOS 1.x al fine di riorganizzarlo ed ottimizzarlo, alla luce dell'esperienza maturata durante la stesura della versione precedente. Nonostante TinyOS 2.x abbia mantenuto molti dei concetti chiave delle precedenti versioni di TinyOS 1.x (primo fra tutti l'architettura componentbased e eventdriven codificata da *NesC*),presenta al momento una implementazione profondamente diversa dalle precedenti realease causandone la non compatibilità delle due versioni.

Ciascuno dei più importanti sottosistemi di TinyOS 2.x è stato associato ad un documento *TEP* (TinyOS Enhancement Proposal) che illustra le linee guida della progettazione e la struttura del sottosistema stesso, facendo riferimento ad implementazioni esemplificative. Ogni *TEP* è aperto a modifiche e revisioni da parte della comunità di utilizzatori.

Le novità introdotte dalla versione 2.x riguardano:

- il processo di inizializzazione e la fase di boot. La sequenza di *Bootstrap* è stata modificata con l'introduzione delle interfacce Boot e Init: il comando *Init.init()* consente di inizializzare con certezza componenti e hardware prima del Boot vero e proprio del sistema. Il suo utilizzo risulta essere

particolarmente utile per assicurarsi che l'inizializzazione sia avvenuta con successo;

- la gestione della coda dei task, lo scheduler del sistema operativo è ora un componente modulare che può pertanto essere sostituito o modificato con facilità;

- la gestione energetica, le funzionalità radio sono normalmente non operative dopo il boot del sistema. L'interfaccia *SplitControl*, che gestisce le comunicazioni wireless, richiede uno start esplicito ed esclusivo. Si evitano in questo modo sprechi energetici, in caso di non utilizzo delle funzionalità radio;

- la struttura architetturale basata su livelli di astrazione. La struttura è stata rivisitata e realizzata dall'*Hardware Abstraction Architecturee* (HAA) (vedi **Capitolo 2.3.1 Hardware Abstraction Architecture**);

- i timer sono ora solo di tipo assoluto: è stato rimosso la possibilità di implementare timer relativi, perchè è concettualmente scorretto reimpostare il clock hardware interno con un valore differente da quello impostato in fase di boot;

- la struttura dei buffer dei messaggi è stata modificata, creando una modalità di accesso alle differenti modalità di comunicazione, come ad esempio radio e seriale, simile, così da favorire la semplicità di programmazione;

- nuovo concetto di distribuzione di servizi paragonabile ad una vera e propria API, semplificando notevolmente lo sviluppo delle applicazioni;

- maggiore flessibilità alle piattaforme hardware realizzata con l'introduzione della versione 1.2 del linguaggio *NesC* e l'utilizzo dell'HAA.

2.3.1 Hardware Abstraction Architecture (HAA)

Nel TinyOS 2.x le risorse fisiche di un dispositivo sono mappate in un *Hardware Abstaction Layer* (HAL) software così da consentire la portabilità e semplificare lo sviluppo dello strato applicativo. Tuttavia questo approccio ha un certo peso sulle prestazioni e non primeggia per efficienza energetica.

Nel contesto delle WSN, dove l'ottimizzazione dei consumi energetici e delle risorse hardware sono requisiti funzionali indispensabili, occorre trovare un compromesso tra le caratteristiche di modularità, flessibilità e riusabilità da un lato e l'utilizzo efficiente delle risorse a disposizione dall'altro Per raggiungere questo scopo, gli sviluppatori di TinyOS 2.x hanno realizzato una *Hardware Abstraction Architecture* (HAA) articolata in tre distinti strati orizzontali di componenti: *Hardware Presentation Layer* (HPL), *Hardware Adaptation Layer* (HAL) e *Hardware Interface Layer (HIL)*. La sovrapposizione di questi tre strati definisce in modo graduale, tra il sistema operativo e le applicazioni, un'interfaccia indipendente dalla piattaforma.

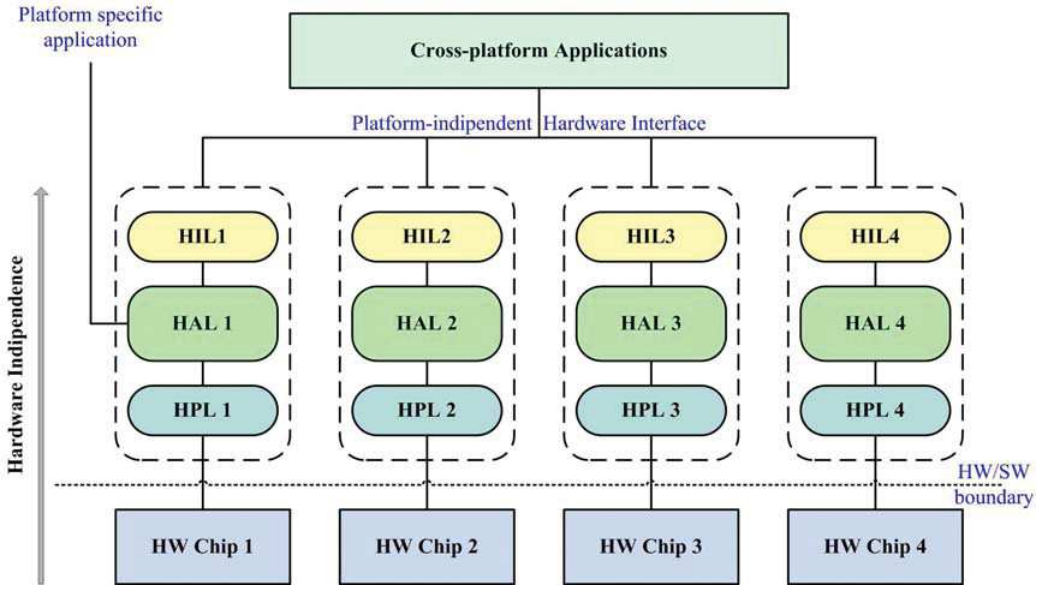


Figura 2.1: Struttura del Hardware Abstraction Architecture

La ragione principale alla base della strutturazione dell'HAA è stata quella di voler accrescere la flessibilità del sistema separando le astrazioni specifiche delle piattaforme hardware, dai componenti wrapper attraverso i quali si realizza l'interfaccia platform-indipendent. Ogni strato è caratterizzato da funzionalità offerte ben definite ed è subordinato alle funzionalità esportate dai livelli inferiori; percorrendo l'HAA dal basso verso l'alto diminuisce la dipendenza dei componenti dall'hardware e aumenta potenzialmente la riusabilità delle applicazioni.

Hardware Presentation Layer

L'Hardware Presentation Layer (HPL) è il livello dell HAA a diretto contatto con i dispositivi hardware dei nodi sensori e presenta le funzionalità e i servizi dello strato hardware ai componenti di livello superiore, integrando i concetti nativi del sistema operativo. La presenza dell'HPL, consente di nascondere la complessità dell'hardware esportando al resto del sistema un'interfaccia usufruibile in modo più semplice, rendendo trasparente all'utilizzatore l'esistenza di gran parte del codice hardware-dipendent e dispensando gli sviluppatori dalle difficoltà legate all'interazione di basso livello con i dispositivi fisici. Ciascun componente dell'HPL è unico e determinato completamente dalle specifiche del modulo hardware di cui rappresenta l'astrazione. Questa caratteristica limita pertanto la libertà d'azione del progettista nell'implementazione dei componenti dell'HPL.

Hardware Adaptation Layer

L'Hardware Adaptation Layer (HAL) rappresenta il nucleo dell'architettura TinyOS 2.x. I componenti di questo livello utilizzano le interfacce di basso livello, esportate dai componenti dell'HPL, per realizzare astrazioni di alto livello che nascondano la complessità tipicamente associata all'utilizzo delle risorse hardware. In contrasto con i componenti dell'HPL, i componenti HAL sono in grado di mantenere informazioni di stato e possono essere impiegati per realizzare arbitraggio e controllo delle risorse. Sebbene questo strato dell'architettura aumenti significativamente il livello di astrazione, resta una forte dipendenza dalle risorse hardware.

Hardware Interface Layer

I componenti dell'Hardware Interface Layer (HIL) costituiscono l'ultimo livello dell'HAA. Tali componenti, partendo dalle astrazioni platform-specific fornite dall'HAL sottostante, esportano allo strato applicativo un'astrazione di ti-

po hardware-indipendent. L'HIL, pertanto, semplifica sensibilmente lo sviluppo e accresce significativamente la portabilitàà delle applicazioni, nascondendo agli sviluppatori le differenze hardware degli strati sottostanti. La complessità dei componenti dell'HIL dipende, in generale, dalla complessità dell'hardware sottostante e dalle funzionalità richieste all'interfaccia platform-indipendent.

2.4 TinyDB

TinyDB[3] è un middleware (costruito al di sopra di TinyOS 1.X) per l'accesso ai dati misurati dai sensori di una rete astraendo allo stesso tempo l'intera rete che viene vista come un database. È composto dalla parte software (sviluppata in *NesC*) eseguita sui singoli sensori e da un'interfaccia client testuale o grafica(sviluppata in Java, attraverso una serie di api Java di natura non molto diversa da JDBC). Diversamente dalle altre soluzioni per processare informazioni in TinyOS, TinyDB non richiede di scrivere codice C per sensori.

L'accesso avviene attraverso un linguaggio dichiarativo molto simile a SQL (chiamato *TinySQL*). Il concetto di query però è lievemente diverso da quello dei DBMS tradizionali: qui la query viene iniettata nella rete e rimane attiva per un tempo specificato. Durante questo periodo i sensori ritrasmettono i loro dati a intervalli di tempo precisi, quindi una query aggiorna i dati in possesso dell'utilizzatore a intervalli costanti invece che una sola volta.
Il sistema è sviluppato per sensori *Motes* sotto forma di applicazione TinyOS sviluppata in *NesC*.

Caratteristiche del TinyDB sono:

- gestione della topologia di rete: TinyDB gestisce la topologia della rete sottostante mantenendo tabelle di routing. Viene costruita una struttura ad albero ad ogni query che permette di inviare all'utilizzatore i dati da tutti i sensori. Viene inoltre gestita la dinamicità della rete in modo trasparente all'utente;

- query multiple: è possibile eseguire più query contemporanee sulla stessa rete; esse vengono eseguite in maniera del tutto indipendente. È inoltre possibile coinvolgere diversi sottoinsiemi dei sensori della rete;

- gestione metadati: il *framework* mantiene un elenco di metadati che caratterizzano i vari tipi di misurazioni che possono essere ottenuti dai sensori;
- query di alto livello: TinyDB utilizza un linguaggio dichiarativo basato su query che descrive il tipo di dati che si vuole, senza richiedere la conoscenza sul come ottenerli.

Il processo di funzionamento ha inizio quando il client invia una query al nodo collegato fisicamente ad esso (tipicamente un *mote* collegato via cavo a un computer). La query a questo punto viene diffusa a tutti i sensori della rete costruendo una struttura di routing ad albero che ha il nodo collegato a un computer come radice. Ogni nodo provvede a rilevare i dati dai propri sensori, propagare le informazioni che arrivano dai nodi a valle ed eventualmente prima di propagare le informazioni aggregarle. Quando tutte le informazioni sono giunte al nodo radice la query termina e viene visualizzata o passata all'interfaccia Java.

I componenti software contenuti sul singolo sensore sono:

- *Sensor catalog*: questo componente tiene traccia del set delle misure disponibili, e di alcune proprietà (ad esempio il sensore padre nella struttura ad albero usata per il routing) che caratterizzano il singolo sensore;
- *Query processor*: questo è il componente chiave durante l'esecuzione di una query, infatti esso ricevere i dati rilevati dall'hardware, e dai sensori figli, aggrega e filtra tali dati e li rispedisce al sensore padre ad intervalli di tempo specificati nella query stessa;
- *Network topology manager*: questo componente fornisce un'interfaccia per interagire con diversi componenti sottostanti per il routing, esso fornisce i comandi per inviare i risultati e gli eventi per ricevere query e risultati.

2.4.1 TinySQL

Il linguaggio TinySQL è composto fondamentalmente da costrutti di selezione, per filtrare i dati e costrutti di aggregazione. Una query eseguita sulla rete non termina necessariamente quando ogni sensore ha inviato i suoi dati una volta, essa può continuare, quindi si possono ricevere dati relativi a diversi *timestep* dello

stesso sensore. La sintassi degli operatori SELECT, FROM. WHERE. GROUP BY, HAVING è identica agli equivalenti del linguaggio SQL. È possibile eseguire particolari operazioni in risposta ad eventi attraverso il costrutto TRIGGER ACTION, tuttavia attualmente ciò è possibile solo in query che non contengono operatori di aggregazione.

Esempio di query in linguaggio TinySQL.

SELECT temp
FROM sensors
WHERE temp > thresh
TRIGGER ACTION SetSnd (512)
EPOCH DURATION 512

Si nota nella prima parte la somiglianza con SQL, viene selezionato il valore di temperatura misurato da tutti i sensori, i dati vengono filtrati secondo una soglia sulla temperatura: quando tale temperatura supera la soglia viene eseguita l'azione *SetSnd* (SetSnd sul client emette un suono della durata di 512 millisecondi) e specifica la durata di ogni timestep di esecuzione della query (ogni sensore invia i dati ogni 512 millisecondi).

Il flusso dell'applicazione tipicamente ha inizio quando il client invia una query al nodo *sink*. I nodi *sink* agiscono come nodi *gateway* (a lungo raggio d'azione e distribuiti accanto ai sensori); un utente specifica al nodo sink di quali informazioni ha bisogno, il *sink* attua il parsing della query e genera una interrogazione che viene poi immessa nella rete. La query viene diffusa a tutti i sensori della rete tramite la costruzione di una struttura di routing ad albero che ha come radice il *sink*; ogni nodo, a questo punto, provvede a rilevare i dati dai propri sensori, aggregare le informazioni che arrivano dai nodi precedenti e progarle a quelli successivi. Una o più risposte sono rinviate indietro dai sensori al *sink* il quale riunisce ed elabora queste risposte prima di fornire il risultato finale all'utente tramite l'interfaccia grafica.

2.5 Considerazioni su TinyDB e TinyOS

TinyDB pone diversi vincoli alle reti di sensori, ovvero una struttura statica dell'albero per il routing (non può variare configurazione durante una query se non in caso di aggiunta di nuovi nodi); inoltre si ha l'impossibilità di spostare la radice della query durante l'esecuzione della stessa. Un altro problema è dovuto al fatto che eseguire query che si prolungano nel tempo potrebbe portare a consumi di energia troppo elevati per i sensori. Di fatti TinyDB prevede un sistema di sincronizzazione, ovvero per query lunghe più di una certa soglia (tipicamente pochi secondi) ogni sensore rimane in stato di elaborazione per gli stessi quattro secondi per ogni periodo di campionamento, mentre per il resto del tempo rimane in stato di sleep. Questo comporta ad un potenziale sovraccarico dei nodi portando ad un consumo non uniforme di energia (i nodi più vicini alla radice trasferiscono più dati e quindi consumano di più).

Altri problemi dipendono invece dal sistema operativo TinyOS 1.x su cui è stato costruito il database. Infatti TinyOS 1.x è stato rimpiazzato dalla versione più recente 2.x creando problemi di compatibilità fra gli applicativi delle due versioni. I sviluppatori di Berkley hanno dunque abbandonato il supporto della versione 1 per dedicarsi esclusivamente a quella attuale, rendendo con il passare degli anni TinyOS 1.x un sistema obsoleto e non pienamente appoggiato in caso di problemi e malfunzionamenti. Per questi motivi il progetto TinyDB è stato attualmente abbandonato. È stata creata d'altro canto una versione non retro-compatibile del database apposita per TinyOS 2.x (TinyDB 2.x), ma quest'ultima non è più sotto licenza open-source e si discosta dall'obiettivo principale di un database aperto e portabile mirato alle reti di sensori, dunque di conseguenza dalle nostre specifiche.

Capitolo 3

SQLite e DB studiati

Con l'abbandono della scelta di utilizzare TinyDB come database principale si è dovuto analizzare una serie di *Relational Database Management System* (RDBMS) con l'obiettivo di trovare quello più adatto alle esigenze richieste da una WSN. Sono stati presi in esame i principali e più utilizzati RDBMS open source attualmente disponibili. Qui di seguito vengono spiegati le caratteristiche principali, i punti di deboli e i punti di forza di ognuno di essi.

3.1 SQLite

"Small. Fast. Reliable. Choose any trhee."

SQLite[4] è una piccola ed efficiente libreria software scritta in linguaggio C che implementa un motore SQL incorporabile all'interno di una qualsiasi applicazione. Il suo creatore è D. Richard Hipp, che lo ha scritto nel 2000 e rilasciato come software Open Source di pubblico dominio, privo di qualsiasi licenza.

Essendo una libreria, non è un processo standalone utilizzabile di per sè ; può essere utilizzata solamente richiamandola da un altro programma implementato in C/C++ oppure in altri linguaggi per cui esistono binding (in particolare Tcl). È integrato nella versione 5 di PHP, consentendogli di non aver più bisogno di appoggiarsi ad un RDBMS esterno come MySQL e divenendo più concorrenziale rispetto alla piattaforma Web ASP.NET di Microsoft. La versione utilizzata per l'implementazione della nostra applicazione è la terza e prende il nome di SQLite3, la quale offre migliorie e nuove funzioni rispetto a quelle precendenti.

SQLite permette di ottenere una base di dati (comprese tabelle, query, form, report) incorporate in un unico file, come il modulo *Access* di *Microsoft Office* o il modulo Base di *OpenOffice.org*, o prodotti specifici come *Paradox* o *FileMaker Pro* rendendo le operazioni di backup di un database molto semplice, trattandosi praticamente di un file. Il vantaggio di avere un database contenuto in un programma è che non sono necessarie nè amministrazione, nè configurazione della rete. Sia il client che il server sono eseguiti insieme nello stesso processo.

Può essere fatto funzionare con 16KB di spazio dello stack e 100KB dell' heap. È pertanto molto diffuso su sistemi dotati di limitate capacità di calcolo, memoria RAM e di memoria per il salvataggio permanente (per esempio cellulari, PDA, lettori MP3 o ,come nel nostro caso specifico, sensori). Un caso tipico sono i sistemi embedded dove le modeste richieste di SQLite lo distinguono dagli altri RDBMS decisamente più esigenti.

Le caratteristiche principali sono:

- le transazioni sono atomiche, consistenti, isolate e durabili (*ACID*) anche in caso di crash di sistema o mancanza improvvisa della corrente;
- non è necessario alcun tipo di operazione di amministrazione per il suo funzionamento, supporta comunque il comando *SQL VACUUM*;
- non necessita di un account per l'accesso ad un database;
- supporta in maniera quasi totale lo standard SQL92;
- ogni database consiste di un unico file il cui formato interno è indipendente dalla piattaforma;
- supporta database che possono essere anche molto grandi; attualmente il limite è 2TB (241 byte);
- i file di database possono essere condivisi tra più macchine;
- è multipiattaforma;
- è molto piccolo (meno di 250KB per l'intera libreria, 150Kb senza le funzionalità aggiuntive. Sqlite3 arriva ad avere dimensioni intorno ai 500Kb);

- codice sorgente di pubblico dominio e ben commentato per facilitare eventuali modifiche ad esso;
- completamente autonomo: non ha dipendenze esterne di nessun tipo;
- è molto veloce; in molti casi più veloce di MySQL e PostgreSQL (se molte operazioni vengono eseguite all'interno di una singola transazione);
- il codice sorgente è liberamente disponibile, chiaro e ben commentato;
- l'API è semplice da utilizzare;
- contiene un programma di utilità che permette l'accesso al database anche manualmente (come MySQL e Postgresql e tanti altri DB) o tramite scripting.

Una peculiarità di SQLite è quella di gestire i tipi di dato in modo molto flessibile: ogni campo può contenere qualsiasi tipo di dato e non siamo obbligati a specificarlo nei campi di una tabella. Indicare dei data type rimane una pratica consigliabile, allo scopo di mantenere la compatibilità con altri database, ma si tratta di un'operazione quasi del tutto ininfluente. Un'altra soluzione per rendere non solo un applicativo portabile su diverse piattaforme e diversi database, ma anche per per salvaguardare la coerenza e la solidità dei dati all'interno del database stesso è quello di utilizzare un'estensione di PHP chiamata *PHP Data ObjectsPDO*.

Ci sono, naturalmente, anche degli svantaggi:

- non possiede stored procedure, gestione dei permessi e molte altre funzionalità tipiche dei colossi;
- non ha una vera gestione della concorrenza (le applicazioni che lo utilizzano, se necessario, devono implementarla);
- non offre alcuna cache per le query (e non ne ha la possibilità, non esistendo un processo server centrale);
- non ha protocolli di rete, non essendo utilizzabile come programma standalone; è possibile utilizzare un database in remoto, ma solo servendosi del filesystem del sistema operativo, con prestazioni difficilmente accettabili;

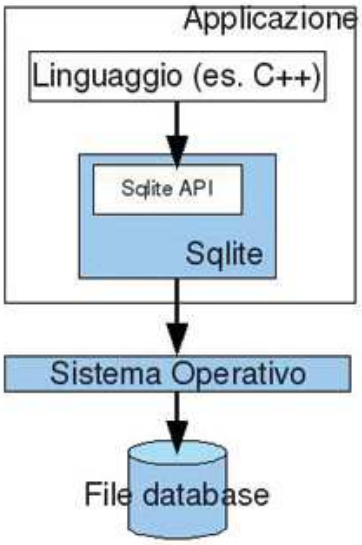


Figura 3.1: Struttura di SQLite

- non è in grado di gestire dati binari in modo sicuro;
- non supporta alcuni importanti comandi SQL (non ha le funzioni *RIGHT* e *FULL OUTER JOIN*, *LEFT OUTER JOIN* è implementato; gestisce i *TRIGGER* ma con alcune limitazioni; il vincolo *FOREIGN KEY* è interpretato ma non c'è l'obbligo di rispettarlo; il comando *ALTER TABLE* consente solo di modificare il nome della tabella o aggiungere colonne alla fine).

3.2 MySQL

MySQL[5] è un database management system (DBMS) relazionale, composto da un client con interfaccia a caratteri e un server, entrambi disponibili sia per sistemi Unix come GNU/Linux che per Windows, anche se prevale un suo utilizzo in ambito Unix. Dal 1996 supporta la maggior parte della sintassi SQL e si prevede in futuro il pieno rispetto dello standard *ANSI*. Possiede delle interfacce per diversi linguaggi, compreso un driver ODBC, due driver Java e un driver per Mono e .NET. Il codice di MySQL è di proprietà della omonima società, viene però distribuito con la licenza GNU GPL oltre che con una licenza commerciale. Fino alla versione 4.0, una buona parte del codice del client era licenziato con la GNU LGPL e poteva dunque essere utilizzato per applicazioni commerciali.

Dalla versione 4.1 in poi, anche il codice dei client è distribuito sotto GNU GPL. Esiste peraltro una clausola estensiva che consente l'utilizzo di MySQL con una vasta gamma di licenze libere.

MySQL è stato progettato e sviluppato tenendo in considerazione la performance come parametro fondamentale. La struttura interna dell'RDBMS sottolinea questo proposito: sono stati sviluppati differenti motori di archiviazione, il cui compito è di gestire i file che contengono, in formato opportuno, le tabelle dei database.

I principali motori realizzano compromessi diversi tra funzionalità e prestazioni. La loro presenza permette all'amministratore di utilizzare il sistema più appropriato in base alle proprie esigenze. Inoltre in MySQL è presente il supporto nativo per il clustering, le repliche master-slave e parallele. Dal punto di vista della compatibilità con lo standard SQL questo RDBMS si può considerare adatto alla maggior parte degli utilizzi comuni; infatti il linguaggio SQL di MySQL comprende numerose estensioni che sono tipiche di altri DBMS, quali PostgreSQL, Oracle e Sybase. In questo modo le query non standard scritte per altri DBMS in alcuni casi funzioneranno senza problemi.

In MySQL una tabella può essere di diversi tipi (o storage engine). Ogni tipo di tabella presenta proprietà e caratteristiche differenti (transazionale o meno, migliori prestazioni, diverse strategie di locking, funzioni particolari, ecc). Esiste poi un'API che si può utilizzare per creare in modo relativamente facile un nuovo tipo di tabella, che poi si può installare senza dover ricompilare o riavviare il server.

Sono i server Web e quelli di posta elettronica che tipicamente non necessitano di funzioni avanzate e possono trarre il massimo beneficio dalla rapidità con cui MySQL risponde, anche in assenza di server dedicati esclusivamente al servizio di database. Non a caso MySQL è il sistema più utilizzato all'interno delle Web farm. Un'architettura dei server opportuna può incrementare notevolmente l'efficienza generale e, al tempo stesso, permettere una gestione semplificata e con un buon livello di sicurezza. La soluzione più diffusa prevede la separazione dei ruoli tra tre cluster: uno dedicato al servizio di frontend, sia esso Web o posta elettronica, uno dedicato alla gestione dei database e infine uno per lo storage. Laddove vengono offerti servizi al pubblico la frequenza con cui si

subiscono attacchi può essere estremamente elevata. La granularità elevata con cui MySQL gestisce i permessi è sicuramente un punto di vantaggio sul fronte della sicurezza.

3.3 PostgreSQL

PostgreSQL[6] è un completo database relazionale ad oggetti rilasciato con licenza libera (stile Licenza BSD). PostgreSQL è una reale alternativa sia rispetto ad altri prodotti liberi come MySQL, Firebird SQL e MaxDB che a quelli a codice chiuso come Oracle, Informix o DB2 ed offre caratteristiche uniche nel suo genere che lo pongono per alcuni aspetti all'avanguardia nel settore dei database.

PostgreSQL usa il linguaggio SQL per eseguire delle query sui dati. Questi sono conservati come una serie di tabelle con chiavi esterne che servono a collegare i dati correlati. La programmabilità di PostgreSQL è il suo principale punto di forza ed il principale vantaggio verso i suoi concorrenti: PostgreSQL rende più semplice costruire applicazioni per il mondo reale, utilizzando i dati prelevati dal database. I database SQL conservano dati semplici in *flat table*, richiedendo che sia l'utente a prelevare e raggruppare le informazioni correlate utilizzando le query. PostgreSQL permette anche agli utenti di definire nuovi tipi basati sui normali tipi di dato SQL, permettendo al database stesso di comprendere dati complessi. Per esempio, si può definire un indirizzo come un insieme di diverse stringhe di testo per rappresentare il numero civico, la città, ecc. Da qui in poi si possono creare facilmente tabelle che contengono tutti i campi necessari a memorizzare un indirizzo con una sola linea di codice.

PostgreSQL, inoltre, permette l'ereditarietà dei tipi, uno dei principali concetti della programmazione orientata agli oggetti. La programmazione del database stesso può ottenere grandi vantaggi dall'uso delle funzioni. La maggior parte dei sistemi SQL permette agli utenti di scrivere una procedura, un blocco di codice SQL che le altre istruzioni SQL possono richiamare. Comunque il SQL stesso rimane inadatto come linguaggio di programmazione, pertanto gli utenti possono sperimentare grandi difficoltà nel costruire logiche complesse.
I punti di forza della programmabilità di PostgreSQL:

- incremento delle prestazioni, in quanto la logica viene applicata diret-

tamente dal server di database in una volta, riducendo il passaggio di informazioni tra il client ed il server;

- incremento dell'affidabilità, dovuto alla centralizzazione del codice di controllo sul server, non dovendo gestire la sincronizzazione della logica tra molteplici client e i dati memorizzati sul server;

- inserendo livelli di astrazione dei dati direttamente sul server, il codice del client può essere più snello e semplice.

Il principale problema di questo database è dovuto alle prestazioni meno brillanti rispetto ad altri database che rendono consigliato il suo utilizzo all'interno di realtà complesse dove si preferisce l'affidabilità rispetto all'efficienza computazionale (si tratta di una condizione frequente nelle grande aziende dove le base dati ha dimensione elevate e i dati in esse contenute hanno valore economico rilevante).

3.4 Motivazioni e considerazioni sulla scelta di SQLite

SQLite è il sistema più performante fra tutti quando si tratta di applicativi che richiedono frequenti operazioni SELECT su un numero di campi e di tabelle limitati. Questo è il caso di un'applicazione dove le operazioni di SELECT per il monitoring in tempo reale dei parametri rilevati dai sensori di WSN sono molto frequenti (nel nostro caso specifico una ogni secondo per ogni sensore collegato).

Altri vantaggi di SQLite sono la sua semplicità e portabilità. Per quanto riguarda la prima può essere inserito, sotto forma di libreria, all'interno di applicazioni stand-alone. Ad esempio in tutti quegli applicativi che non necessitano di un database centralizzato e condiviso. In questo caso la compattezza di SQLite permette un'installazione semplice dell'applicativo in quanto non richiede la configurazione di database esterni. Trattandosi di un unico file, il backup e la rimozione del database gestito da SQLite è altrettanto semplice. Per quanto la portabilità è frequente l'utilizzo all'interno dei software di librerie di astrazione di database che permettono all'applicativo di interfacciarsi a una molteplicità di

RDBMS diversi in maniera trasparente rispetto al funzionamento dell'applicativo stesso (questo grazie soprattutto all'utilizzo del sistema PDO).

In base a tali considerazioni è stato scelto SQLite come RDBMS per lo sviluppo di questa applicazione. Questo vantaggio in termini di flessibilità è purtroppo compensato da una limitazione sui comandi SQL che possono essere utilizzati, che sono pari al sottoinsieme delle funzioni supportate da PostgreSQL e MySQL. Comunque nonostante questa limitazione, per le funzioni e operazioni necessarie in questo progetto, quelle fornite da SQLite sono state sufficienti per la realizzazione del progetto.

Capitolo 4

Tecnologie usate

4.1 Texas Instruments EZ430-RF2500

Il dispositivo utilizzato per sviluppare la rete di sensori wireless e il sistema basato su database per l'interrogazione della stessa è prodotto dalla Texas Instruments e si chiama *EZ430-RF2500*[7]. La Texas Instruments Incorporated, meglio conosciuta nel mondo della industria elettronica con l'acronimo TI, è una compagnia statunitense, con sede a Dallas in Texas, famosa per lo sviluppo, la produzione e la vendita di dispositivi elettronici a semiconduttori e di tecnologia informatica in genere. La TI, preceduta dalla Intel e dalla Samsung, risulta attualmente essere il terzo produttore mondiale di dispositivi elettronici e possiede centri di produzione, sviluppo e commercializzazione in 3 diversi continenti.

Il dispositivo EZ430-RF2500 è costituito da una piccola scheda elettronica (*target board*) nella quale sono collegati i due integrati fondamentali della piattaforma: la *CPU MSP430* e il *Transceiver CC2500*[8]. L'EZ430-RF2500 può essere programmato tramite l'apposito adattatore USB, utilizzato inoltre per la comunicazione tramite il connettore UART (comunicazione di tipo seriale). L'alimentazione è fornita dall'adattatore stesso, prelevata direttamente dalla porta USB. L'EZ430-RF2500 nel caso in cui non sia connesso direttamente al Personal Computer, può essere alimentato tramite due batterie stilo tipo "AAA", tramite l'apposito contenitore. Sulla target board sono collocati, oltre ai due integrati principali, due led di colore rosso e verde, collegati ad altrettante uscite digitali del microcontrollore.

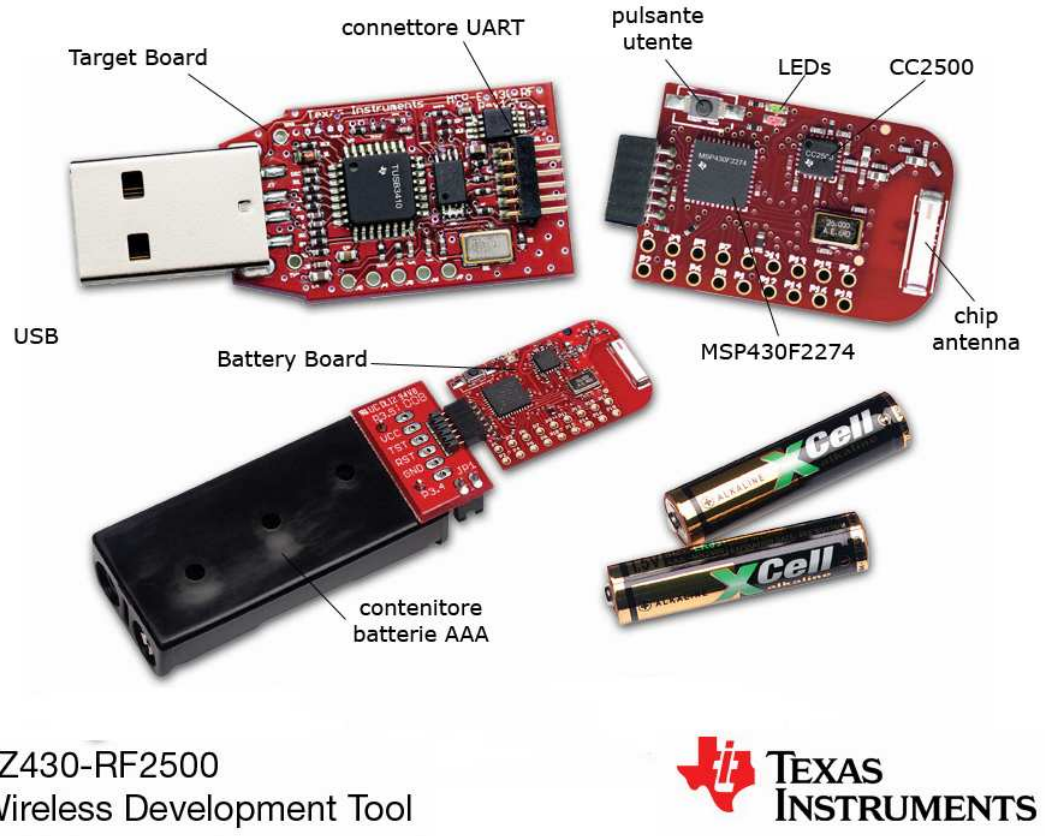


Figura 4.1: EZ430-RF2500

Sono inoltre disponibili ulteriori 18 pin per connettere altrettante porte di I/O analogiche e digitali. È inoltre disponibile un pulsante a pressione per l'interazione con l'utente. Le funzionalità radio sono garantite dal transceiver grazie alla presenza di un antenna a 2.4 GHz integrata nel circuito stampato.

Peculiarità di questo dispositivo è l'utilizzo di un protocollo di rete a bassissimo consumo e sviluppato dalla stessa Texas Instruments: *SimpliciTI*. Il dispositivo non fa uso di un vero e proprio sistema operativo, ma il codice del firmware (in linguaggio C standard) è sviluppato e compilato tramite l'ambiente *IAR Embedded Workbench* e caricato sul nodo, utilizzando l'apposita interfaccia USB.

4.1.1 Il microcontrollore MSP430

L'EZ430-RF2500 monta la versione MSP430F2274 del microcontrollore MSP430[9] che dispone di 32KB di memoria Flash programmabile, per ospitare il codice dei programmi e 1 KB di RAM, per l'esecuzione degli stessi. Il consumo di energia di questi dispositivi è veramente ridotto e ogni singolo dispositivo ha una gamma di periferiche analogiche e digitali con un bus di comunicazione on-chip all'interno del chip stesso. Questo alto livello di integrazione e di una sofisticata struttura di interrupt rendono questi dispositivi adatti a una buona progettazione. La potenza di elaborazione fornita dai processori varia da 8 a 25 MIPS.
Nello specifico l'MP430 include:

- una CPU RISC a 16 bit interconnessa tramite un bus dati (*MDB*);
- un bus indirizzi (*MAB*) a 16 bit con le memorie di sistema (RAM e Flash/ROM);
- un controller DMA;
- un moltiplicatore hardware;
- periferiche I/O digitale;
- Watch Dog timer;
- general purpose volte;
- timer in real time;
- USART che può essere configurato per UART;
- comunicazione I2C o SPI;
- ADC e DAC;
- controllore LCD;
- scansione I/f e modalità di emulazione dedicate.

I dispositivi sono mappati in memoria e collegati all'indirizzo di sistema e ai bus di dati attraverso un adattatore che regola il bus dati a 8 bit di larghezza. I bus sono classici indirizzi di memoria e il bus di memoria dati ed i dispositivi sono

mappati in memoria. Il modello del programma di memoria è quello di uno spazio lineare. La area superiore è assegnata alla Flash/ROM (pari a 128KB) e ai vettori di interrupt e quelle inferiori per la maggior parte sono per registri di funzioni speciali e dispositivi a 8 e 16 bit. Il sistema di clock è stato specificatamente pensato per le applicazioni che richiedono l'alimentazione a batteria.

4.1.2 Il transceiver CC2500

Si tratta di un transceiver prodotto dalla Chipcon per la Texas Instruments, a basso costo e consumo energetico; è capace di funzionare con tensioni di alimentazione che vanno da un minimo di 1.8V a un massimo di 3.6V quindi adatto per applicazioni alimentate a batteria. Funziona nel range di frequenza ISM (*Industrial, Scientific e Medical*) e per applicazione a piccolo raggio di azione (SRD). La banda di frequenza è 2400-2483.5 MHz ed è basata sulla sintesi diretta della frequenza di base. Supporta diverse modalità di modulazione del segnale radio e raggiunge una velocità di trasmissione massima di 500 kbps. Per l'accesso ai buffer dei dati e di configurazione è disponibile un'interfaccia SPI a 4 pin.
Caratteristiche:

- dimensioni ridotte: 4 x 4 mm, 20 pins di connessione;
- alta sensibilità: -101 dBm a 10 kbps;
- basso consumo di corrente: 13.3 mA in RX a 250kbps;
- funzionamento a multi-canale;
- buffer in modalità FIFO da 64Byte, sia in ricezione che in trasmissione;
- informazioni RSSI, ovvero sulla potenza del segnale radio ricevuto;
- Wake-on-radio, il dispositivo può attendere la ricezione di un frame in uno stato di sleep a basso consumo, passando nello stato attivo per l'effettiva ricezione del frame.

Le caratteristiche sopra descritte rendono il CC2500 un ottimo dispositivo per l'implementazione in un nodo sensore di una WSN.

4.2 Protocollo SimpliciTI

SimpliciTI[10] è un protocollo di rete open source proprietario sviluppato da Texas Instruments dedicato ad applicazioni a radiofrequenza a basso consumo (*Low Power RF*) e bassi costi poichè richiede poche risorse di sistema (tipicamente il protocollo utilizza meno di 4-8KB di Flash per la memorizzazione del codice e meno di 512-1024B di RAM per il funzionamento). Funziona nella banda ISM a 2.4GHz ed è rivolto a reti con un numero massimo dispositivi pari a 256, capace di comporre topologie di reti a stella e peer to peer, ma non reti mesh.

SimpliciTI supporta la cifratura a chiave simmetrica dei dati; di fatto include un software di sicurezza per la crittazione dei messaggi in grado di assicurare la sicurezza dei dati trattati della rete. è implementata anche la migrazione fra diversi canali (chiamata anche *Frequency Agility*) cioè la capacità dei dispositivi facenti parti della rete di spostarsi sui vari canali trasmissivi al fine di evitare la possibilità che differenti messaggi entrino in conflitto tra loro distruggendo il contenuto informativo. Vantaggio principale di questo protocollo è la bassa quantità di risorse richieste in confronto ai protocolli visti precedentemente.

4.2.1 Architettura del protocollo

Si può dire che SimpliciTI, concettualmente, abbia un'architettura a tre livelli. Il livello più alto di applicazione è l'unico programmabile dall'utilizzatore. I tre livelli che compongono lo stack protocollare di SimpliciTI sono:

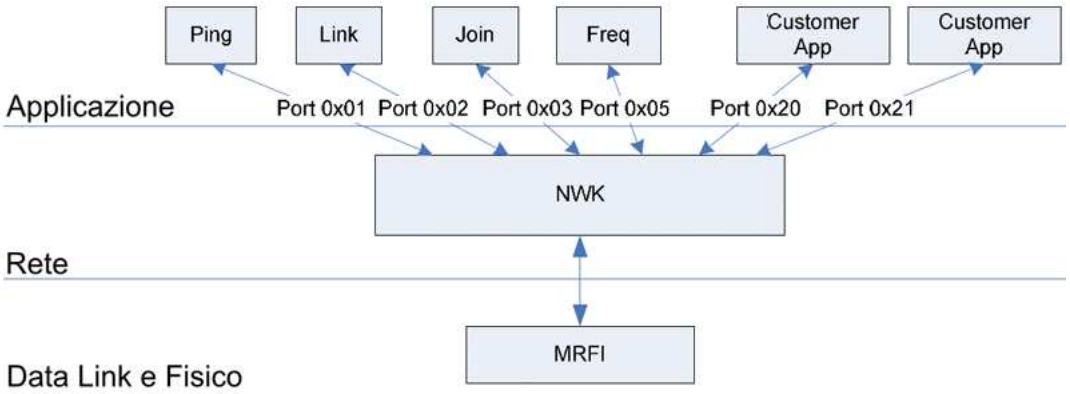


Figura 4.2: Architettura dei livelli del protocollo SimpliciTI

Livello applicazione
Fornisce alcune funzioni su determinate porte per gestire l'interazione con la rete. Il programmatore può sviluppare proprie applicazioni utilizzando le due porte disponibili mediante l'utilizzo delle API fornite dal protocollo. Le applicazioni dell'utente sono in grado di accedere alle funzionalità dei livelli sottostanti come invio e ricezione dei messaggi, inizializzazione e configurazione della rete.

Livello di rete
Il livello di rete ricalca le caratteristiche del network layer del modello ISO/OSI. Esso è regolamentato da una serie di parametri che vengono interpretati e inclusi nel codice al momento della compilazione. Alcuni di essi posso essere modificati a livello applicazione tramite l'interfaccia *ioctl()*. Suddetti parametri possono rappresentare le seguenti caratteristiche:

- valore della frequenza base per la trasmissione;
- numero di frequenze supportate e parametri generali del modulo radio;
- metodo di modulazione del segnale;
- data rate;
- supporto al cambio di canale (frequency agility);
- chiavi di crittazione;
- indirizzo dei nodi;
- controlli generici sulla rete.

Livello fisico
È gestito dall'interfaccia MRFI (*Minimal RF Interface*) ed è il livello più basso in termini d'astrazione. Si occupa del collegamento tra il livello di rete e l'hardware a disposizione. L'MRFI è sviluppato per funzionare su diversi dispositivi, a seconda del transceiver utilizzato si potrebbe avere la necessità di avere differenti implementazioni del codice, tuttavia questo livello rimane sempre inalterato per ogni tipo di modulo radio impiegato.

4.2.2 Dispositivi logici

Il protocollo SimpliciTI definisce tre tipologie di dispositivi logici collegabili alla rete:

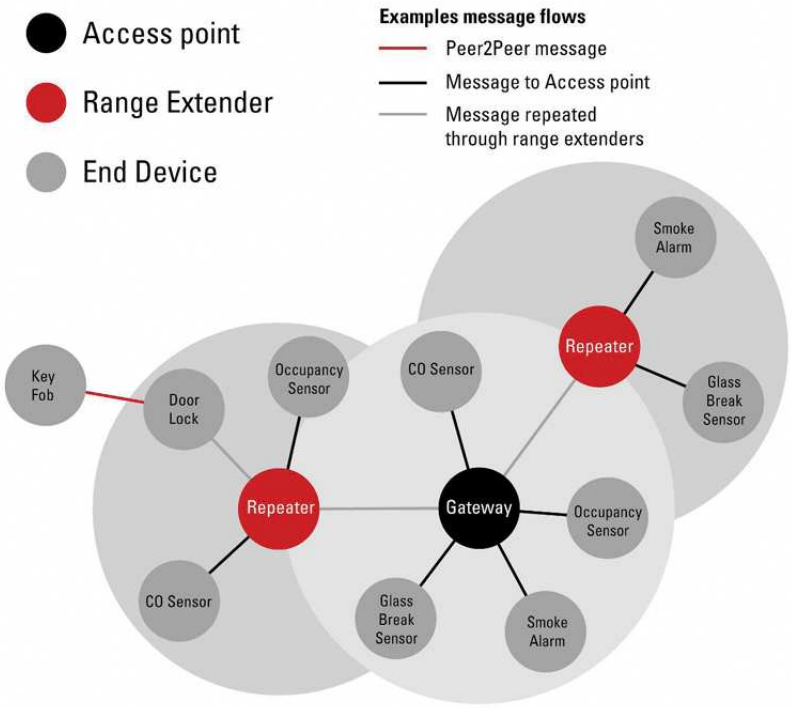


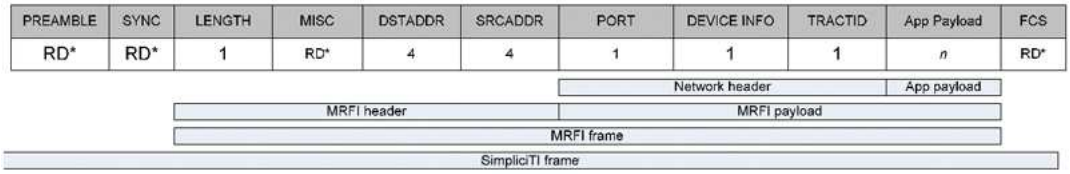
Figura 4.3: Esempio di rete con protocollo SimpliciTI

- **End Device (ED)** che svolge la funzione di nodo sorgente e di nodo sensore ed è alimentato di solito da batterie. Ogni rete ne implementa almeno uno e può essere attivo solo in trasmissione oppure in trasmissione/ricezione. Inoltre può essere sempre attivo o dormiente, attivandosi solo per la comunicazione;
- **Access Point (AP)** che svolge di fatto il ruolo di coordinatore della rete e può essere anche interconnesso ad altre reti (ovvero fungere da Gateway). Non ne è obbligatoria la presenza. Tra le funzioni dell'AP vi sono: la gestione degli indirizzi di rete e la memorizzazione e inoltro dei messaggi agli ED dormienti. Un AP può implementare anche funzionalità tipiche degli ED (sensori e attuatori) o svolgere il compito del RE.

- **Range Extender (RE)** che svolge la funzione di ripetitore e permette di ampliare il campo d'azione della rete. È un particolare tipo di nodo sempre attivo e generalmente alimentato direttamente. Il dispositivo trasmette unicamente i messaggi ricevuti.

4.2.3 Frame di trasmissione

Ogni frame trasmesso col protocollo SimpliciTI è suddivisibili in tre porzioni, in base al livello dello stack del protocollo che lo genera.



Field	Definition	Comments
PREAMBLE	Radio synchronization	Inserted by radio HW
SYNC	Radio synchronization	Inserted by radio HW.
LENGTH	Length of remaining packet in bytes	Inserted by FW on Tx. Partially filterable on Rx.
MISC	Miscellaneous frame fields	Differ for different radios. May be absent.
DSTADDR	Destination address	Inserted by FW. Filterable depending on radio.
SRCADDR	Source address	Inserted by FW.
PORT	Forwarded frame (7), Encryption context (6) Application port number (5-0).	Inserted by FW. Port namespace reserves 0x20-0x3F for customer applications and 0-1F for NWK management.
DEVICE INFO	Sender/receiver and platform capabilities	Inserted by FW. Details below.
TRACTID	Transaction id	Inserted by FW. Discipline depends on context. Need not be sequential.
APP PAYLOAD	Application data	$0 \leq n \leq 50$ for non-802.15.4 radios; $0 \leq n \leq 111$ for 802.15.4 radios
FCS	Frame Check Sequence	Usually a CRC appended by the radio hardware.

Figura 4.4: Frame del protocollo SimpliciTI

4.3 PHP

Per la realizzazione della nostra applicazione Web è stato utilizzato il linguaggio PHP[11]. PHP (acronimo ricorsivo di *PHP Hypertext Preprocessor*, preprocessore di ipertesti) è un linguaggio di scripting interpretato, con licenza open source e parzialmente libera (ma incompatibile con la GPL), originariamente concepito per la realizzazione di pagine Web dinamiche. Attualmente è utilizzato principalmente per sviluppare applicazioni Web lato server ma può essere usato anche per scrivere script a linea di comando o applicazioni standalone con interfaccia grafica.

La versione utilizzata per il progetto è la 5 alla quale sono successivamente state aggiunte le estensione che permettono di interfacciarsi molto agevolmente alle funzionalità della libreria di SQLite3 (*sqlite3.dll*, la versione di PHP5 include inizialmente solo la prima versione standard di SQLite) e della libreria PDO (*pdo_sqlite.dll*).

4.3.1 PHP Data Objects (PDO)

PDO sta per *PHP Data Objects* ed è un nuovo sistema per interfacciarsi a database con PHP 5. Grazie a driver specifici per ogni tipo di database, PDO è in grado di offrire funzionalità ed interfacce standard a prescindere dal tipo del database con cui si interagisce. Il sistema PHP Data Objects si compone di due classi, una per interfacciarsi al database e per interrogarlo (PDO), un'altra per accedere al record set restituito dalla query (*PDOStatement*). L'utilità della programmazione ad oggetti è che è possibile estendere una classe in qualsiasi momento, rendendola più adatta alle nostre esigenze.

Parte II

Progetto e implementazione dell'applicazione

Capitolo 5

Progetto

La progettazione del software è passata attraverso una serie di fasi; per prima cosa sono stati raccolti i requisiti da sviluppare. Successivamente sono stati sviluppati alcuni casi d'uso e *activity diagram* allo scopo di chiarire la struttura delle classi principali. Il tutto è stato creato seguendo le regole dello standard ANSI/IEEE 830 relative all'SRS (*Software Requirements Specification*) e a quelle che definiscono l'UML (*Unifical Modelling Language*).

Inizialmente il software era stato progettato solamente per la monitorizzazione in tempo reale e l'archiviazione di parametri ambientali di una WSN tramite un'applicazione Web; successivamente a tale applicazione è stata aggiunta la possibilità di archiviare anche documentazioni, articoli e tesi inerenti alle reti di sensori.

5.1 Requisiti funzionali

Di seguito sono elencati i requisiti principali che hanno portato alla realizzazione del software.

RF01 - Monitorizzazione dati del sensore

- *Descrizione:* questa funzione permette tramite l'*homepage* di monitorizzare in tempo reale i vari parametri (temperatura, segnale, batteria) rilevati da tutti i sensori collegati in quel momento all'Access Point.
- *Input:* voce del menù *sensori*.

- *Processo:* il programma effettua una ricerca nel database *dbsensori.sqlite* sulla tabella *sens* prelevando solamente la tupla con la data più recente per ogni sensore collegato.

- *Output:* vengono visualizzate le ultime rilevazioni più recenti fatte dai sensori.

RF02 - Visualizzazione dati dalla tabella reporting

- *Descrizione:* questa funzione permette di visualizzare archivio storico contenuto nella tabella reporting salvata sul server all'interno del database *dbsensori.sqlite*.

- *Input:* voce del menù *tabella reporting*.

- *Processo:* il programma effettua una ricerca su tutte le tuple contenute nel database *dbsensori.sqlite* sulla tebella *archivio*.

- *Output:* vengono visualizzati i vari dati sui sensori contentuti nell'archivio storico con la rispettiva data di acquisizione.

RF03 - Login/logout

- *Descrizione:* un utente può accedere al sito Web tramite la procedura di login, inserendo un *username* e una *password*, decise dall'amministratore al momento della sua registrazione nel database. Il Logout permette la disconnessione dell'utente. Tutte le procedure presentate di seguito potranno essere effettuate solo a seguito del login.

- *Input:* username e password, pulsante "logout" accessibile dal menù.

- *Processo:* viene fatto un controllo nel database per verificare la presenza di quella password in corrispondenza di quel determinato utente.

- *Output:* in caso di riconoscimento positivo si passa alle pagine protette del sito. In caso negativo viene scritta la causa di errore. Quando viene premuto il pulsante di disconnessione la schermata torna nella *homepage*.

I requisiti RF04 e RF05 sono accessibili solamente da un utente registrato che abbia effettuato l'accesso tramite il login. Si noti che un utente può svolgere solo funzioni di inseriremento di file, ma non di eliminazione.

RF04 - Visualizzazione e ricerca di documentazioni/articoli/tesi

- *Descrizione:* l'utente una volta registrato può visualizzare l'intero archivio delle documentazioni/articoli/tesi contenuti sul server. Può anche tramite un *form* e un tasto *cerca* cercare specifici documenti a partire dal titolo, gruppo, autore o relatore (solamente nel caso delle tesi).
- *Input:* un *form* e un tasto *cerca*, voce nel menù.
- *Processo:* il programma effettua una ricerca nel database *Tesi* , *Articoli*, *Documenti* a seconda della voce nel menù selezionata e nel caso si tratti di una ricerca di uno specifico documento viene selezionata la tupla corrispondente.
- *Output:* Vengono visualizzati i documenti con i rispettivi dati.

RF05 - Inserimento di documentazioni/articoli/tesi

- *Descrizione:* una volta effettuato l'accesso al sistema possono essere inserite nuove documentazioni, tesi o articoli nel database principale. È solamente necessario selezionare il *.pdf* da caricare e il sistema provvederà in automatico a compilare i dati di immissione aggiornando il database relativo al documento uplodato.
- *Input:* pulsante "upload" dopo aver selezionato il documento dal file system.
- *Processo:* il documento *.pdf* viene caricato sul server, mentre i dati relativi all'utente vengono prelevati dalla tabella *utente* contenuta nel database *database.sqlite* e utilizzati per compilare i campi necessari per l'aggiornamento del database relativo al file caricato.
- *Output:* messaggio di corretto caricamento o di errore.

I requisiti RF06 e RF07 sono accessibili solamente dall'amministratore del sito.

RF06 - Inserimento nuovo utente

- *Descrizione:* questa funzione permette di inserire un nuovo utente nel sistema. Deve essere scelto un *username* e una *password* sicura (quest'ultima verrà codificata tramite l'algoritmo crittografico MD5) che verrà utilizzata dal nuovo utente per accedere al sistema.

- *Input:* compilazione dei relativi campi tramite *form* contenuta nel database *database.sqlite*.

- *Processo:* il nuovo utente viene inserito nella tabella *utente* .

- *Output:* messaggio di corretto caricamento o di errore.

RF07 - Cancellazione di un utente/documentazione/articolo/tesi

- *Descrizione:* questa funzione permette di eliminare un utente , una documentazione, un articolo o una tesi dai rispettivi database.

- *Input:* titolo del documento/articolo/tesi o lo username dell'utente da eliminare tramite un *form*; pulsante *cancella*.

- *Processo:* viene ricercata all'interno del rispettivo database la tupla corrispondete al nome immesso nel *form* e viene cancellata da quel database.

- *Output:* messaggio di corretta cancellazione o di errore.

5.2 Use Case Diagram

Qui di seguito viene mostrato lo studio dei casi d'uso di un utente che accede all'applicazione Web.

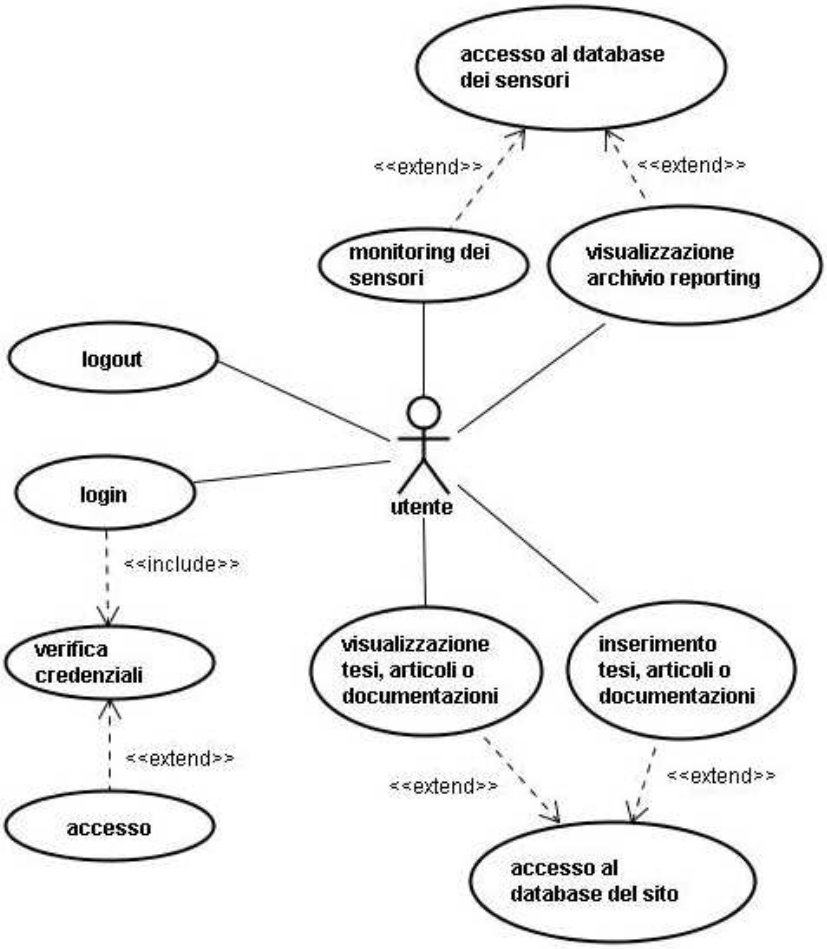


Figura 5.1: Caso d'uso di un utente nel dettaglio

5.3 Scenario: monitoring sensori o visualizzazione dell'archivio

In questo paragrafo verrà descritto a parole l'esempio di scenario che si verifica più frequentemente durante l'utilizzo dell'applicazione ovvero la monitorizzazione dei parametri rilevati dai sensori in tempo reale o la consultazione dell'archivio contenente la cronologia dei dati raccolti fino ad ora.

Le azioni svolte dall'utente sono relativamente semplici: l'utente accede alla homepage dell'applicazione (*SensiNet.php*) e il sistema automaticamente esegue una ricerca sui dati contenuti nella tabella *sens* (situata in *dbsensori.sqlite*)

visualizzando in tempo reale le ultime rilevazioni prese da ognuno dei sensori collegati. Nel caso venga invece selezionata la voce del menù *Tabella reporting* il sistema caricherà una nuova pagina (*SensiNet_rep.php*) dove verrà visualizzato l'archivio storico, prendendo i dati direttamente dalla tabella *archivio* (situata in *dbsensori.sqlite*). In qualsiasi momento l'utente può tornare dalla pagina *SensiNet_rep.php* alle operazioni di monitoring attraverso la voce del menù *Sensori*.

5.3.1 Caso d'uso in forma tabellare

Verrà mostrati ora la scena appena descritta suddivisa in due casi d'uso in forma tabellare: *UC1* si riferisce alla monitorizzazione dei sensori, *UC2* si riferisce alla visualizzazione della tabella reporting.

UC: UC1.
Nome: Monitoring sensori.
Attori: Utente.
Precondizioni:

L'utente si è collegato all'homepage dell'applicazione.

Sequenza eventi:

- Il sistema carica la pagina *SensiNet.php*.
- Il sistema stabilisce una connessione con il database *dbsensori.sqlite*.
- Il sistema tramite un interrogazione SQL seleziona la tupla con la data più recente dalla tabella *sens*. Questo per ogni sensore collegato al'AP.
- Il sistema visualizza i risultati all'utente tramite la pagina Web.
- Il sistema ricarica ogni 3 secondi la pagina mantenendo la connessione aperta sul database ma interrogando nuovamente la tabella *sens* in cerca di tuple più recenti.

Sequenza alternativa:

L'utente seleziona dal menù la voce *Tabella reporting*. Questo porta al caso d'uso UC2.

Postcondizioni:

In qualsiasi momento l'utente potrà disconnettersi dall'applicazione. In tal caso verrà interrotta le connessione con il database.

UC: UC2.
Nome: Visualizzazione archivio reporting.
Attori: Utente.
Precondizioni:

L'utente si è collegato all'homepage dell'applicazione.

Sequenza eventi:

- L'utente ha selezionato dal menù la voce *Tabella reporting*.
- Il sistema carica la pagina *SensiNet_rep.php*.
- Il sistema stabilisce una connessione con il database *dbsensori.sqlite*.
- Il sistema tramite un interrogazione SQL seleziona tutte le tuple contenute nella tabella *archivio*.
- Il sistema visualizza i risultati all'utente tramite la pagina Web.

Sequenza alternativa:

L'utente seleziona dal menù la voce *Sensori*. Questo porta al caso d'uso UC1.

Postcondizioni:

In qualsiasi momento l'utente potrà disconnettersi dall'applicazione. In tal caso verrà interrotta le connessione con il database.

5.3.2 Activity Diagram

Questo Activity Diagram analizza i casi d'uso UC1 e UC2. Sono stati inseriti all'interno dello stesso diagramma con lo scopo di dare una visione più generale e completa dello scenario.

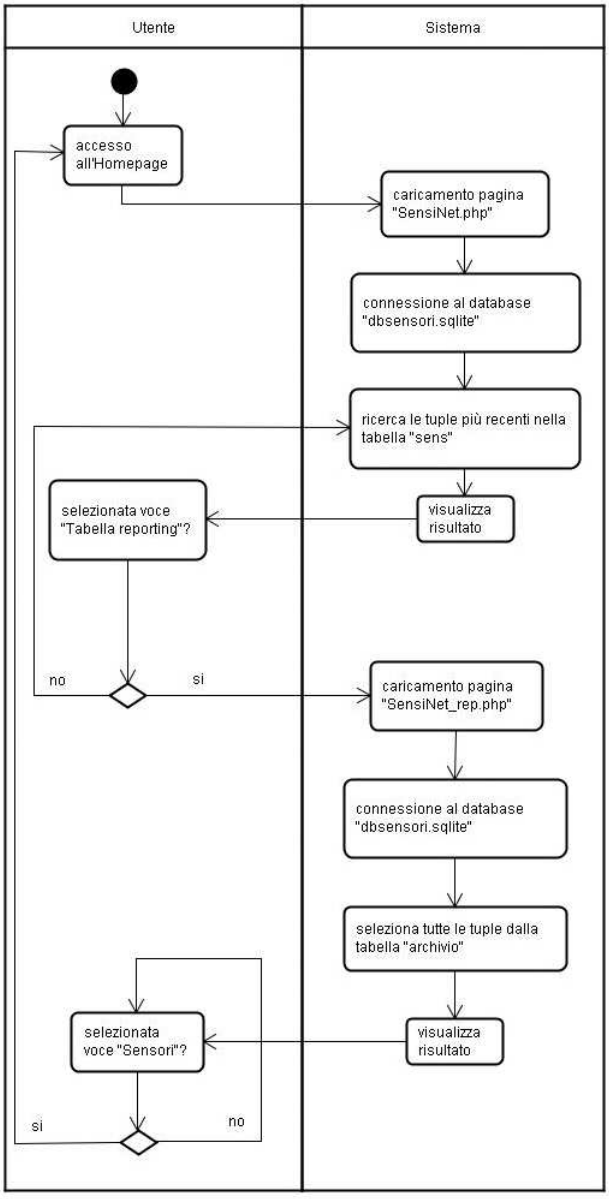


Figura 5.2: Activity Diagram per il monitoring sensori e visualizzazione archivio

5.4 Struttura dei database

Il sistema si appoggia su due database distinti: *dbsensori.sqlite* e *database.sqlite*. La scelta di utilizzare due file-database diversi è stata principalmente quella di tenere separato tutto ciò che riguarda i sensori da quello che riguarda la gestione del sito.

Figura 5.3: Organizzazione e struttura dei database

Il database *dbsensori.sqlite* contiene tutte le informazioni riguardanti i sensori e si suddivide in due tabelle. La prima, chiamata *sens*, contiene i parametri ambientali e di stato rilevati dai dispositivi collegati in quel momento all'access point. Questa tabella viene continuamente aggiornata dal programma wrapper che fa da tramite tra l'AP e il database; allo stesso tempo viene consultata dall'applicazione Web per il monitoring in tempo reale degli ultimi parametri rilevati.

La seconda tabella è chiamata *archivio* e contiene la media giornaliera delle informazioni contenute nella tabella *sens*. Il motivo dell'utilizzo di questa tabella è di ottenere risposte sull'andamento dei sensori nell'arco del tempo in maniera più chiara e in tempi decisamente ridotti rispetto alla consultazione della stessa

tabella *sens*. Ogni volta che la tabella *archivio* viene aggiornata tutte le tuple di *sens* vengono eliminate. Il motivo di questo riazzeramento è dovuto ad un rispiarmo di spazio fisico sul server. Infatti sulla tabella *sens* vengono scritte, tramite il wrapper, più rilevazioni al secondo per ogni dispositivo connesso; questo comporta nell'arco di una giornata ad un numero elevato di tuple aumentando le dimensione del database in maniera sempre più graduale. Riazzerando ogni volta la tebella stilandone la media giornaliera e salvandola in un'unica tupla nell'archivio si evita questa problematica.

Il database *database.sqlite* contiene invece tutte le informazioni riguardanti la gestione del sito: dalla documentazione presente, all'elenco e gestione degli utenti.

La documentazione è organizzata in tre tabelle: *Articoli*, *Documenti* e *Tesi*; ognuna per ogni rispettiva tipologia di documento (*.pdf*) che può essere uplodato all'interno del sito. Il campo *link* presente in ognuna delle tabelle contiene il *path* dove si trova effettivamente il documento all'interno del file system del server e verrà utilizzato per recuperare il documento una volta che l'utente ne richiede la consultazione. Per quanto riguarda la gestione degli utenti vengono utilizzate altre due tabelle: *utente* e *authlogin*. Nella prima sono contenute le informazione di ogni utente registrato al sito. La prima tupla della tabella è riservata all'amministratore il quale è l'unico ad avere permessi di scrittura su di essa. Nella tabella *authlogin* invece viene scritta dinamicamente una tupla per ogni sessione creata a seguito di un login effettuato dall'utente. Una volta che quest'ultimo si scollega dal sito (tramite la funzione logout o chiudendo il *browser*) la sessione viene cancellata.

5.4.1 Schemi logici

Elenco di tutti gli schemi logici di ogni tabella.

SENS (address, temperatura, batteria, segnale, data)

ARCHIVIO (address, temperatura, batteria, segnale, data)

ARTICOLI (ID_articoli, titolo, autore, gruppo, link, data)

DOCUMENTI (ID_doc, titolo, autore, gruppo, link, data)

TESI (ID_tesi, titolo, autore, relatore, gruppo, link, data)

UTENTE (ID_utente, nome, cognome, relatore, correlatore, gruppo, username, password, ruolo, telefono, email)

AUTHLOGIN (authcode, idutente)
FK: idutente REFERENCES utente

Capitolo 6

Implementazione

6.1 Visione generale

Nel seguente capitolo si tratterà l'architettura e il funzionamento dell'applicazione per il monitoraggio e l'archivio di dati di una Wireless Sensor Network. Lo scopo principale è quello di poter visualizzare in tempo reale i parametri ambientali rilevati da una rete di sensori tramite una pagina Web in php e poterli successivamente archiviarli in un database contenente la media delle misurazioni effettuate nell'arco di una giornata. Il sistema si basa sull'utilizzo di SQLite, un *Data Base Management System* (DBMS) Relazionale sia per la visualizzazione che per la memorizzazione dei dati rilevati.

L'obiettivo primario è l'interazione dell'utente al più alto livello possibile di astrazione. La pagina Web risulta dunque essere l'elemento fondamentale che interfaccia l'utente con il sistema. Essa fa uso delle librerie specifiche di PHP (*sqlite3.dll* e *pdo_sqlite.dll*) per rendere possibile la comunicazione con il database, mentre l'interfacciamento tra questo e la WSN avviene tramite un *wrapper* scritto in linguaggio C++.

Le query usate hanno una struttura *SQL-like*, ovvero basata su una sintassi simile allo standard SQL. Il vantaggio nell'utilizzo di un modello di interrogazione generica, indipendente alla struttura fisica della rete sottostante, rende il sistema potenzialmente in grado di gestire flussi di dati provenienti da sorgenti eterogenee e quindi totalmente svincolato dalle piattaforme hardware presenti aumentando la portabilità del software.

Oltre alle attività di monitoraggio e archiviazione sono state aggiunte altre funzionalità attorno alla pagina Web. Di fatto è stato creato un sito in PHP il quale dovere è quello di raccogliere ogni tipo di documentazione riguardanti le WSN creando così un archivio consultabile da chiunque sia interessato a progettare o ampliare progetti già esistenti all'interno dell'ambito universitario. Le varie documentazioni saranno accessibili e uplodate direttamente dall'utente il quale dovrà essere registrato al sito per poter interagire con l'elenco completo dei documenti. Questi sono stati suddivisi in tre grandi categorie: documentazione, articoli e tesi. L'unico formato supportato è il formato *.pdf*. La struttura del sito è stata scritta in HTML e la grafica è gestita da un foglio di stile esterno (CSS).

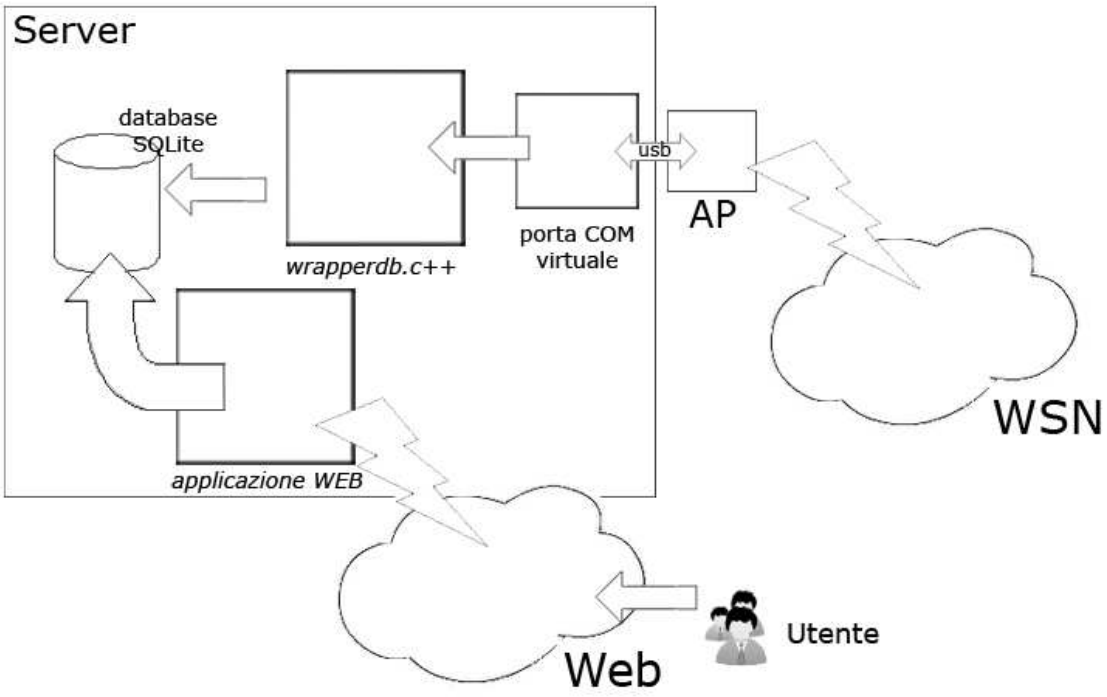


Figura 6.1: Sistema complessivo

6.2 Software Wrapper

Il Wrapper è costituito fisicamente da un calcolatore e da un EZ430-RF2500 collegato tramite un adattatore USB. La connessione avviene utilizzando una porta seriale "virtuale" (COM), creata dal driver software del dispositivo TUSB3410

"EZ430-RF USB Debugging Interface" . Quest'ultimo permette, tramite la connessione USB al calcolatore, l'interfacciamento e l'alimentazione di un sensore EZ430-RF2500, utilizzando l'apposito connettore a 6 pin. L'EZ430-RF2500 collegato all'interfaccia Usb, sarà di seguito chiamato Access Point (AP), poichè eseguirà un software apposito, differente da quello in esecuzione sugli EZ430-RF2500 utilizzati come sensori distribuiti nell'ambiente. Essi sono chiamati End Device (ED) e trasmetteranno i dati rilevati nella rete tramite comunicazione radio. La comunicazione dell'AP, tramite porta COM, avviene verso un il software definito *wrapperdb.cpp* scritto in linguaggio c++. La sua funzione sarà quella di gestire la comunicazione tra il database SQLite e la WSN.

6.2.1 Funzionamento del codice

Il codice del software *wrapperdb.cpp* è stato scritto in linguaggio C++, tramite l'ambiente di sviluppo e compilazione eclipse (versione 3.42). Il programma è basato su un ciclo senza fine *while (true)* {...} il quale rimane continuamente in ascolto sulla porta COM a cui è virtualmente collegato l'AP gestendo così la ricezione e la memorizzazione dei dati in arrivo dalla rete di sensori. Una volta ricevuti i dati corretti dalla porta seriale vengono memorizzati direttamente nella tabella *sens* contenuta nel database *dbsensori.net*.

Nello specifico come prima cosa viene creato un *handle* relativo all'apertura della porta COM in lettura (*hComm*) e viene stabilita la connessione con il database tramite il comando *Open* fornito dalle API di SQLite. Una volta all'interno del ciclo principale *while* viene eseguita la funzione *Readfile()* di lettura dalla porta seriale. Questa è una funzione "bloccante", ovvero ritorna il controllo solo alla ricezione del numero di Byte specificato. Nel caso in esame il payload della stringa, contiene 22 byte, così suddivisi:

- 2B per i caratteri di inizio stringa $ e fine stringa #;
- 4B per l'indirizzo sensore;
- 3B per la potenza segnale radio;
- 6B per la temperatura rilevata;
- 3B per lo stato batteria (voltaggio);

- 4B per i caratteri separatori.

Una volta che la stringa dati è stata correttamente ricevuta viene passata alla funzione *explode()*, la quale elabora i dati e genera in uscita un'array di stringhe chiamato *strdati* contente le informazioni ricevute (di fatto questa funzione è simile all'analoga funzione *explode* di PHP). L'array viene poi passato ad una seconda funzione chiamata *prepareQuery* la quale prepara la query che verrà inviata al database *sens* tramite le API fornite da SQLite.
La query creata di fatto sarà strutturata in questa maniera:

```
INSERT INTO sens (address,temperatura,batteria,segnale,data) VALUES ('ADDR','-XX.XC','V.C','RSI','date')
```

dove *ADDR* è l'indirizzo del sensore ed è con contenuto in *strdati[0]*, *-XX.XC* è la temperatura rilevata contenuta in *strdati[1]*, *V.C* lo stato della batteria e *RSI* la potenza del segnale, contenuti rispettivamente in *strdati[2]* e *strdati[3]*. *date* è invece la data esatta in cui la rilevazione è stata compiuta (calcolata in secondi passati dal giorno 01/01/1970).

Come ultima cosa il software effettua un controllo sul tempo ed ogni 86400 secondi trascorsi dalla sua esecuzione (ovvero 24 ore) viene invocata la funzione *aggiornaArchivio()* la quale tramite la query

```
Select ROUND(AVG(AVG(temperatura),1), ROUND(AVG(batteria)), ROUND(AVG(segnale), 1) from sens
```

calcola la media delle acquisizioni dei sensori effettuate nella giornata e chiama la funzione *outSqliteAVG* che costruisce la query da inviare al database *archivio* in maniera simile alla funzione *prepareQuery* con la differenza che la data è formattata in "mm/gg/aaaa". Dopodichè tramite una DELETE * viene cancellata l'intera tabella *sens*.

Per una consultazione del codice completo si rimanda all'**Appendice 8.1**.

6.3 Software AP

La comunicazione del wrapper verso l'EZ430-RF2500 adibito ad Access Point verso la WSN collegata, avviene tramite la porta COM virtuale creata dal driver dell'interfaccia USB.

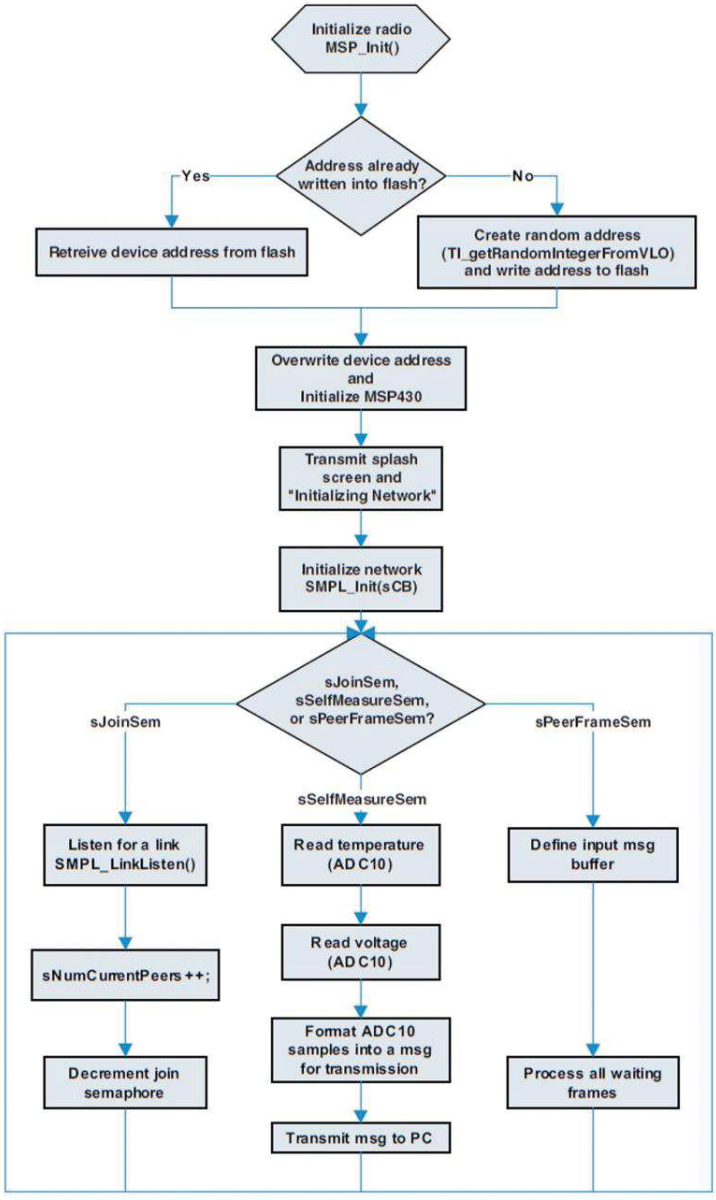


Figura 6.2: Diagramma di flusso del software AP

L'inizializzazione del dispositivo all'avvio avviene tramite le funzioni *BSP_Init()*, che si occupa di inizializzare la comunicazione tra il microcontrollore MSP430

e il transeiver CC2500 e la funzione *MCU_Init()* che nello specifico configura la modalità di funzionamento del MSP430. In particolare viene settata una frequenza di funzionamento di 8Mhz, viene abilitato un interrupt *Timer_A* eseguito a un intervallo di 1 secondo e viene abilitata la comunicazione seriale tramite interfaccia (USCI) UART, configurando un velocità di 9600 Baud in RX/TX. Nella prima inizializzazione i sensori, sia AP che ED, generano e memorizzano nella memoria flash interna un indirizzo casuale, utilizzato per l'identificazione univoca del dispositivo nella rete. Questo indirizzo hardware viene inoltre utilizzato per riconoscere gli ED, momentaneamente disconnessi o riavviati, permettendo all'AP, di riassegnare il precedente indirizzo virtuale (*LinkID*), utilizzato per la comunicazione col protocollo SimpliciTI.

I diversi rami di esecuzione del programma principale sono condizionati da appositi semafori: *sPeerFrameSem*, *aJoinSem* e *sSelfMeasureSem*. Il semaforo *sPeerFrameSem* è attivato quando avviene la ricezione di un frame radio da un ED già connesso all'AP. Il semaforo è incrementato dall'apposita funzione *sCB*, invocata tramite interrupt dal protocollo di rete, quando un frame è in attesa di ricezione. La ricezione effettiva del frame è delegata al ramo di esecuzione condizionato dal semoforo, che al suo termine viene riportato a zero. Il frame ricevuto dall'ED, contiene le informazioni sul dato di temperatura rilevato e sul voltaggio della batteria. Prima di inoltrare questa informazione alla porta seriale, viene completata con l'informazione sulla qualità del segnale, ottenuta grazie a un'apposita funzione del protocollo SimpliciTI. I dati vengono poi formattati in un'apposita stringa e spediti alla porta seriale.

Il ramo di esecuzione condizionato dal semaforo *aJoinSem*, anch'esso attivato dalla funzione *sCB*, viene eseguito nel caso in il frame ricevuto sia il primo frame di connessione di un nuovo ED. In questo caso viene lanciata dall'AP la funzione *SMPL_LinkListen()* che ritorna il *linkID*, con cui effettuare le successive comunicazioni con gli End Device.

L'ultimo semaforo implementato è *sSelfMeasureSem*, esso ha il compito di stabilire il momento in cui effettuare la rilevazione della temperatura da parte dell'AP e di inviarla come stringa alla seriale in modo del tutto analogo a quello degli ED. *sSelfMeasureSem* è attivato dal *Timer_A*, con una frequenza di una volta al secondo.

Le modifiche attuatate al software standard AP per sviluppo dell'applicazione sono state quella di escludere la modalità estesa di visualizzazione dei dati rilevati e cambiare l'unità di misura della temperatura. Esistono di fatto due formati dell'output dati ed in default la temperatura è in fahrenheit:

- *verboseMode*, la modalità estesa (settata in default dal software).
  Esempio di output:
  Node:XXXX,Temp:-XX.XC,Battery:X.XV,Strength:XXX%,RE:no;

- *output_short*, la modalità compatta.
  Esempio di output:
  $*ADDR,-XX.XC,V.C,RSI,N#*.

Il software Wrapper è stato progettato in modo da accettare come input solamente l'*output_short*, dunque è stata esclusa la funzione di invio della modalità estesa dal software standard AP settando il carattere *verboseMode* a 0. È stato anche riconfigurato l'output da $\backslash r\backslash n\$ADDR,-XX.XC,V.C,RSI,N\#$ a $\$ADDR,-XX.XC,V.C,RSI,N\#$ elimando i caratteri speciali $\backslash r$ e $\backslash n$. Per quanto riguarda la conversione da fahrenheit a celsius è stato settato il carattere *degCMode* a 1.

Per una consultazione del codice completo si rimanda all'**Appendice 8.2.1**.

6.4 Software ED

Gli End Device rappresentano i veri e propri nodi della rete di sensori. Essi sono collegati in modalità a stella ad un solo Access Point, al quale comunicano i dati.

Il *joining* con l'AP a distanza di rilevamento è effettuato all'accensione del dispositivo. Il numero massimo di ED collegabili allo stesso AP è fissata dal protocollo di rete SimpliciTI a 30, è un valore comunque modificabili intervenendo sulle costanti che lo definiscono. La portata massima di comunicazione radio può essere estesa tramite l'utilizzo di sensori appositamente programmati per questo scopo, essi prendono il nome di *Range Extender* (RE). Il RE legge i frame ricevuti e li reinoltra sulla rete a maggiore potenza.

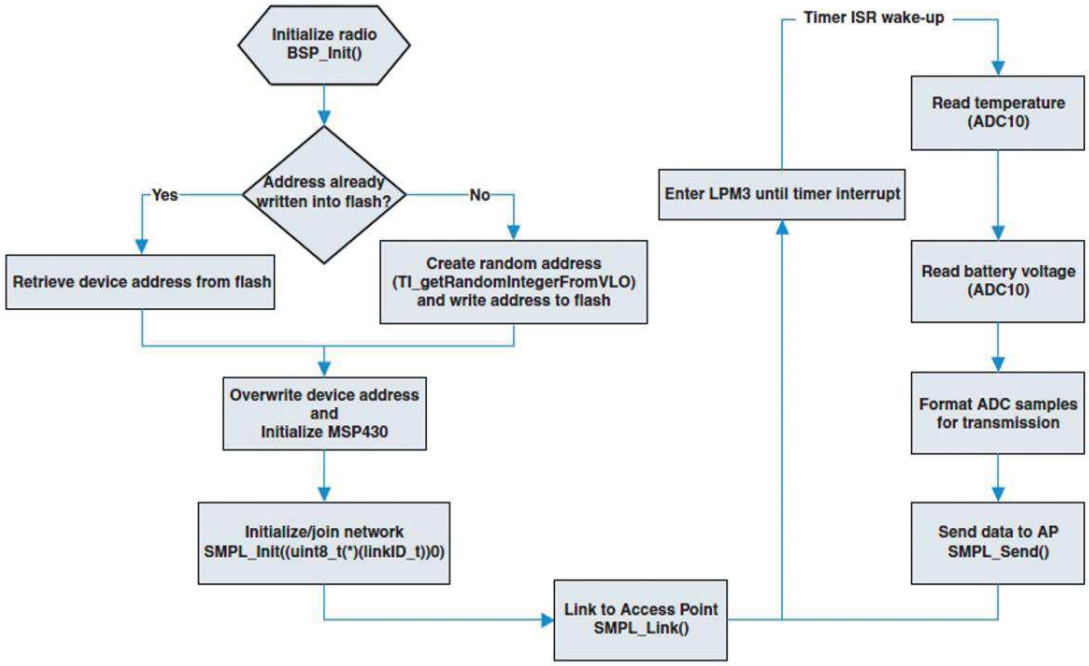


Figura 6.3: Diagramma di flusso del software ED

Dopo l'inizializzazione hardware del dispositivo tramite *BSP_Init()*, viene creato e assegnato l'indirizzo hardware se non è già presente, esattamente come accade per l'AP. Per quanto riguarda invece l'inizializzazione della rete, vi è una sostanziale differenza, in quanto alla funzione *SMPL_Init()* non è associata alcuna funzione di *callback* (sCB) avente il compito di gestire la ricezione dei frame radio. Infatti la comunicazione dell'ED è unidirezionale in trasmissione verso l'AP.

L'End Device si trova per la maggior parte del suo tempo di esecuzione in una modalità a basso consumo (*LPM3*), con spegnimento della cpu e delle funzionalità radio. Sarà il *Timer_A* ad ogni sua esecuzione ciclica una volta al secondo, a riattivare il dispositivo, riportandolo nella modalità a consumo normale e con la trasmissione radio abilitata. Alla riaccensione del nodo vi è la rilevazione della temperatura e dello stato della batteria, l'invio all'AP del messaggio contente queste informazione e il ritorno alla modalità *LPM3*. A differenza del software AP, non sono state apportate modifiche nell'ED.

Per una consultazione del codice completo si rimanda all'**Appendice 8.2.2**.

6.5 Interfaccia Web

Il contenuto dell'interfaccia Web è stata realizzata tramite il linguaggio di markup HTML e per la formattazione è stato adoperato un foglio di stile CSS (*stilehome.css)*. Le applicazioni contenute sono state realizzate utilizzando il linguaggio di scripting PHP5 esteso per le librerie SQLite3 e PDO.

L'applicazione Web, insieme al sito, si suddivide in diverse pagine PHP, ognuna delle quali ha specifiche funzioni. Solamente *SensiNet.php* e *SensiNet_rep.php* riguardano l'applicazione per il monitoraggio e archivio dei dati ricevuti e si appoggiano sul database *dbsensori.sqlite*; mentre gli altri file.php riguardano la gestione del sito e si appoggiano sul database *database.sqlite*.

Le pagine che compongono il sito e l'applicazione sono:

- *SensiNet.php*;
- *SensiNet_rep.php*;
- *login.php*;
- *sensori.php*;
- *home.php*;
- *cerca.php*;
- *Admin.php*;
- *inserimento.php*;

Analizziamo ora in maniera generale ogni singolo file.php rimandando al codice contenuto nell'**Appendice 8.3.1-8** per un'analisi più chiara e completa.

6.5.1 SensiNet.php

È la pagina principale dell'applicazione dove l'utente può visualizzare in tempo reale lo stato e le informazioni prelevate dai sensori di una WSN. La pagina viene ricarita ogni secondo permettendo l'aggiornamento dei valori più recenti rilevati dai dispositivi. Da qui è possibile accedere al sito tramite la voce del menù *Login* (il link è diretto alla pagina *login.php*) o visualizzare tramite la voce *Tabella reporting* (*SensiNet_rep.php*) l'archivio dove sono immaganizzate di giorno in giorno tutte le medie dei parametri dei sensori.
Non è richiesto il login per accedere a questa pagina.

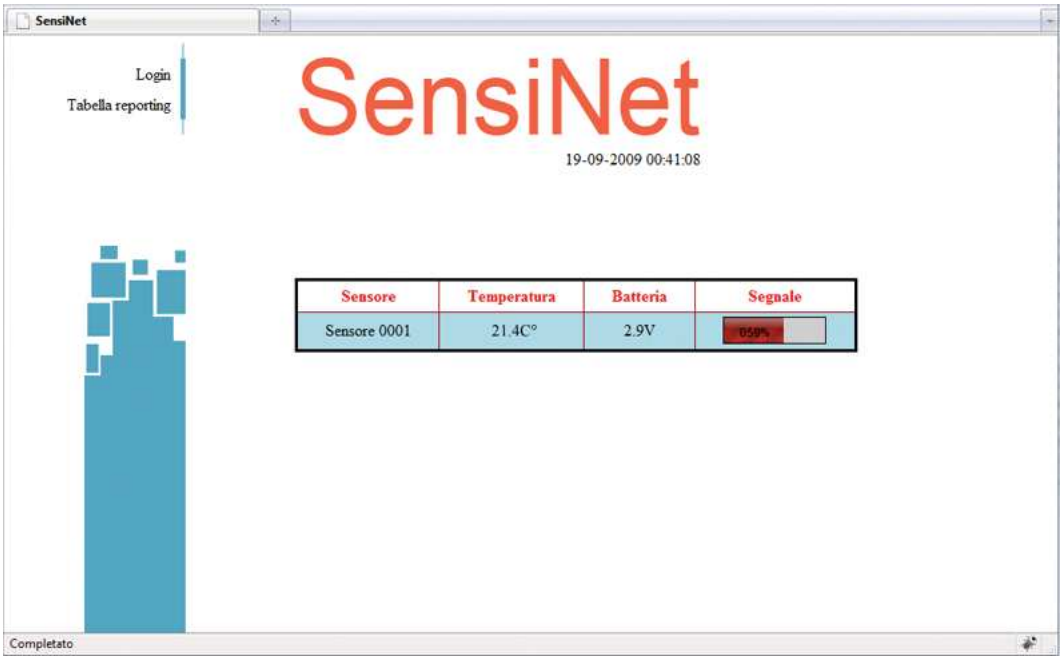


Figura 6.4: Esempio della pagina SensiNet.php

6.5.2 SensiNet_rep.php

Questa pagina viene richiamata ogni volta che un utente desidera visualizzare l'archivio dati delle medie dei valori giornalieri acquisiti dai sensori. La tabella visualizza, a differenza della monitorizzazione in tempo reale, tutti i valori salvati e non solo gli ultimi rilevati in ordine di tempo. A seconda della pagina da cui si è acceduti si ha la possibilità di effettuare il login al sito o, nel caso sia già stato

effettuato, si può ricercare tesi, articoli e documentazioni all'interno del sito dalle rispettive voci nel menù. È possibile inoltre tornare in qualsiasi momento alla monitorizzazione in tempo reale tramite la voce nel menù *sensori* o eseguire il logout dal sito tramite la voce *logout*.

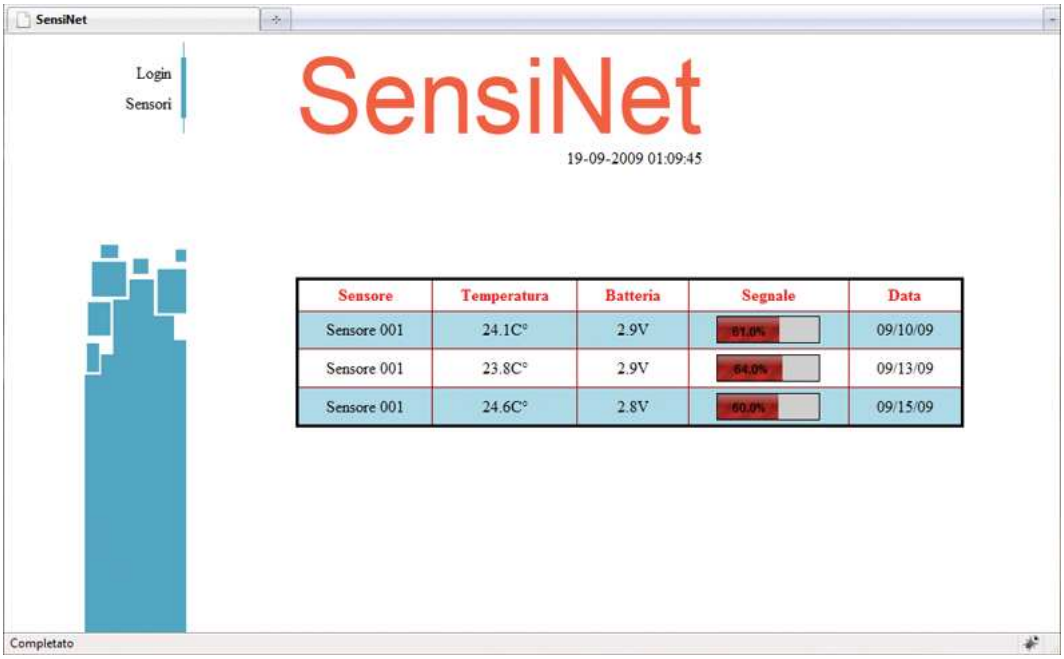


Figura 6.5: Esempio della pagina SensiNet_rep.php

6.5.3 login.php

Il login utente funziona sfruttando le sessioni php. Per inizializzarle all'avvio della pagina di login viene utilizzata la funzione *session_start()*. Per effettuare il login è necessario inserire i dati di autenticazione (*username* e *password*) forniti dall'amministratore del sito. Il form autenticativo invia tramite post i dati alla pagina stessa che nelle prime righe di codice ne controlla il contenuto e in caso di identità verificata effettua il login dell'utente. Una volta effettuato l'accesso ci sarà un reindirezzamento alla pagina *home.php*. I dati relativi all'identità dell'utente rimangono memorizzati nella tabella *utente*.

Il login dell'utente consiste nella creazione (o nell'aggiornamento) di una stringa alfanumerica nella tabella di autenticazione *authlogin*. Il login viene mantenuto tramite una variabile *token* che persiste nella sessione e viene confrontata

con la controparte salvata nel database. Il logout consiste nell'eliminzione del *token*.

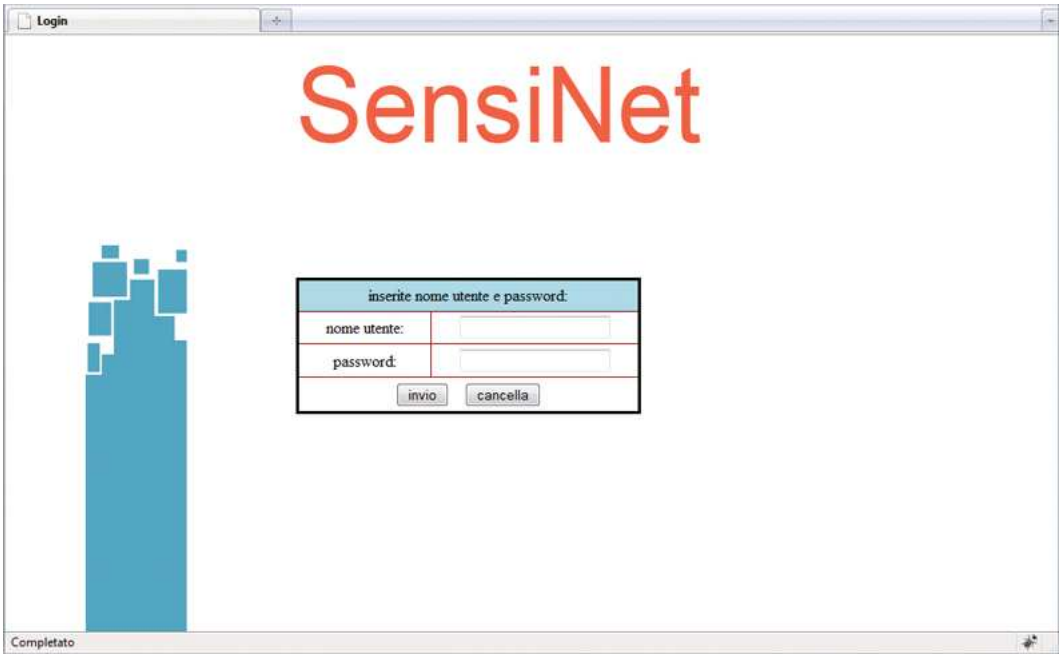


Figura 6.6: Esempio della pagina login.php

6.5.4 sensori.php

La pagina *sensori.php* ha le stesse funzionalità della pagina principale *SensiNet.php*; l'unica differenza è che l'utente è loggato al sito. Di conseguenza attraverso il menù si può visualizzare i vari tipi di documenti (tesi, articoli, documentazione), cosa che non è possibile se non si è loggati. Si può passare in qualsiasi momento alla visualizzazione della tabella reporting o eseguire il logout dal sito.

6.5.5 home.php

Questa è la pagina principale del sito accessibile solo dopo aver effettuato il login. Da qui, oltre alle normali funzioni di monitoring e consultazione dell'archivio dei sensori, è possibile visualizzare tutta la documentazione presente sul database

del sito. Si può accedere alle varie tipologie di documento tramite le apposite voci nel menù. Le tipologie sono:

- tesi, accessibile dalla voce *Tesi*; salvati nella tabella *tesi*;
- articoli, accessibile dalla voce *Articoli*; salvati nella tabella *art*;
- documentazioni, accessibile dalla voce *Documentazioni*; salvati nella tabella *documenti*.

Tutte le tabelle si trovano all'interno del database *database.sqlite*.

È possibile tramite un tasto "cerca" cercare specifici documenti tra quelli mostrati. Una volta cliccato il tasto verrà caricata la pagina *cerca.php* contenente le procedure di ricerca e visualizzazione dei risultati. Un utente inoltre può uplodare nuovi documenti all'interno delle rispettive sezioni, ma non modificare o cancellare quelle già esistenti. Per caricare un nuovo documento basta cliccare sul bottone "upload" e si verrà automaticamente indirizzati alla pagina *inserimento.php* dove verranno svolte le procedure di caricamento di un file. Infine è possibile eseguire il logout dal sito tramite l'apposita voce nel menù.

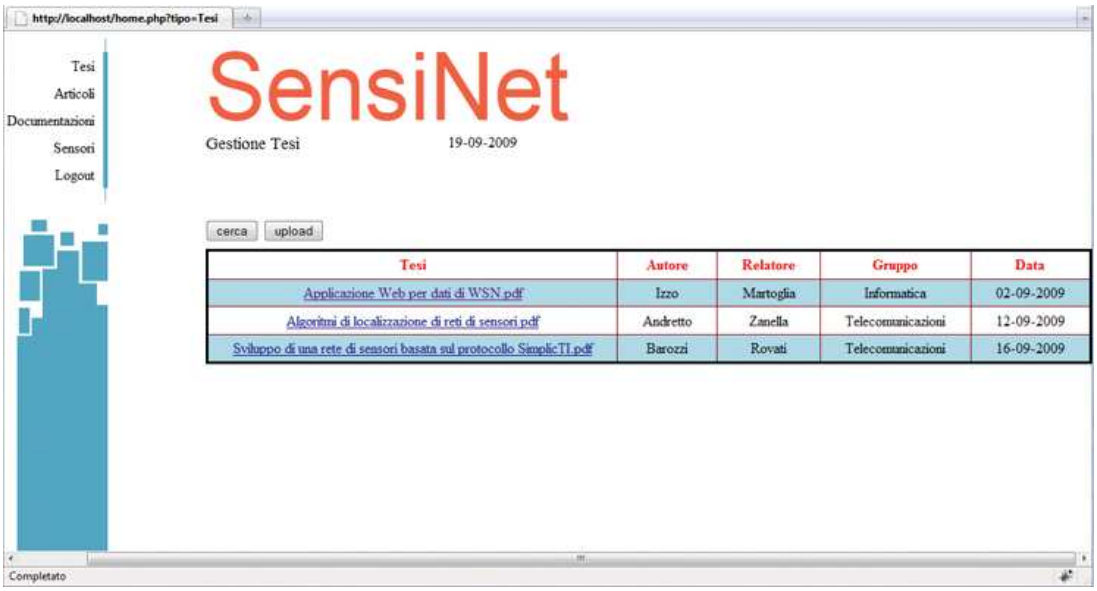


Figura 6.7: Esempio di visualizzazione delle tesi nella pagina home.php

6.5.6 cerca.php

Nella pagina *cerca.php* sono contenute le procedure di ricerca di uno specifico documento a partire dal titolo, gruppo, autore o relatore (solamente nel caso di un documento di tipo tesi). La pagina mostra un *form*, dove verrà inserito il campo da cercare, e un bottone "cerca" il quale interrogherà la tabella in cui è stata richiesta la ricerca selezionando solamente le tuple che contengo il termine scritto nel *form*. Questa pagina si fa carico anche di visualizzare i risultati ottenuti.

6.5.7 Admin.php

Questa pagina è riservata all'amministratore del sito e può accederci solamente tramite *username* e *password* specifici. Da questa pagina è possibile visualizzare la lista completa di ogni documento caricato da ogni utente e, nel caso lo si voglia, di cancellare un documento desiderato. Per prima cosa bisogna accedere alla tabella tipologia del documento dalla voce corrispettiva del menù (*Gestione Tesi*, *Gestione Articoli* o *Gestione Documenti*), poi bisogna inserire tramite un *form* il titolo del documento da rimuovere e cliccare il pulsante "cancella". Dopodichè viene richiamata la pagina *inserimento.php* la quale contiene le procedure per la rimozione del documento. Stesso procedimento può essere ripetuto per la lista completa degli utenti registrati con l'unica differenza che nel *form* va inserito lo *username* dell'utente da cancellare.

L'amministratore oltre a cancellare può anche inserire un nuovo utente accedendo ad una tabella da compilare attraverso la voce del menù *Gestione Utente* e cliccando il bottone "nuovo utente". Verrà dunque visualizzata la tabella contenente i vari campi da completare con le varie informazioni riguardanti l'utente da inserire. Queste infine verrano mandate tramite il bottone "conferma" alla pagina *inserimento.php* che provvederà ad inserirlo nella tabella *utente* del database. È possibile come sempre eseguire il logout dal sito in qualsiasi momento tramite l'apposita voce nel menù.

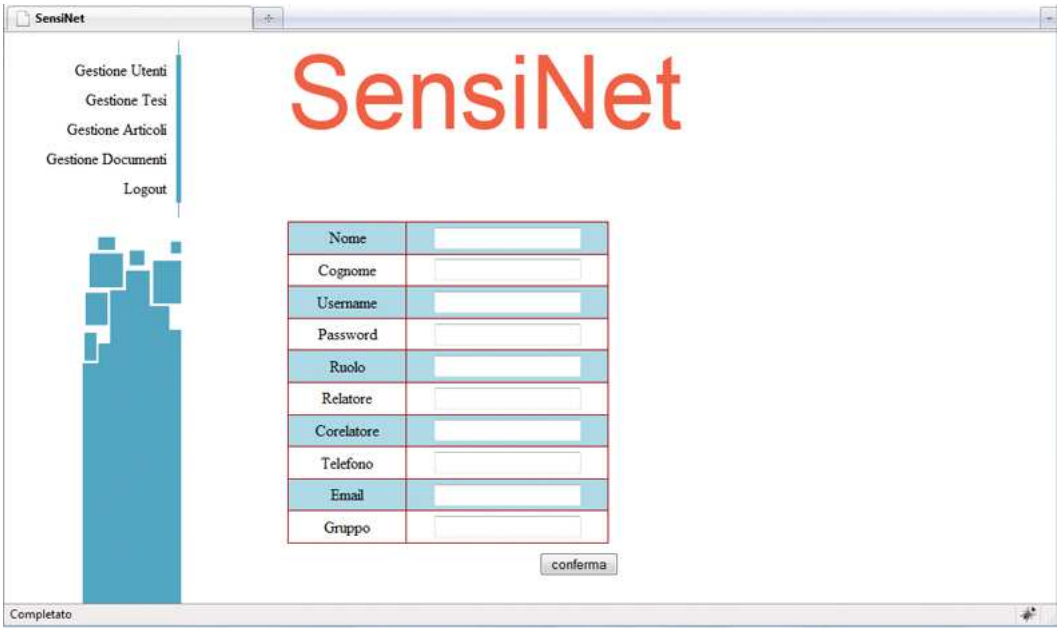


Figura 6.8: Esempio della tabella inserimento utente della pagina Admin.php

6.5.8 inserimento.php

La pagina *inserimento.php* contiene varie procedure richiamate dalle altre pagine Web. Tra queste ci sono le procedure per l'upload di un documento da parte di un utente e anche l'inserimento stesso di un nuovo utente all'interno della tabella *utente*.

Per il caricamento di un file, l'unica estensione supportata per il corretto funzionamento del sito è il *.pdf*. L'utente dopo aver selezionato il documento dal proprio file system clicca sul pulsante "upload" e la pagina provvederà a trasferire il file selezionato nell'apposita directory del server e inserirà in automatico una tupla contente i dati del nuovo documento nella rispettiva tabella, andando a prelevare dalla tabella *utente* le informazioni rigurdanti l'autore e il gruppo di lavoro.

Per quanto riguarda invece l'inserimento di un nuovo utente, vengono prese tramite il metodo POST, le informazioni compilate dall'amministratore al momento della registrazione e viene aggiornata la tabella *utente* con le informazioni sul nuovo utente.

Infine la pagina contiene anche le funzioni per eliminare un utente, una documentazione, un articolo o una tesi dai rispettivi database; viene ricercata all'interno del rispettivo database la tupla corrispondete al nome passato alla pagina immesso precedentemente nel *form* e viene cancellato il risultato ottenuto da quel database.

6.6 Prove sperimentali

Si è scelto di utilizzare SQLite come database per il progetto grazie a suoi numerosi vantaggi tra cui la sua semplicità e portabilità, ma soprattutto per la velocità e le dimensioni ridotte sul disco.

Una serie di test sono stati eseguiti per valutare le prestazioni di Sqlite confrontate a quelle di altri database quali PostgreSql e MySql. L'ambiente di test è stato predisposto su una piattaforma Athlon a 1.6GHz , 1GB di memoria, un disco IDE e sistema operativo RedHat Linux 7.2.

Operazioni	Sqlite	Sqlite (nosync)	PostgreSql	MySql
1.000 INSERT	13.061	0.223	4.373	**0.114**
25.000 INSERT (in una transazione)	0.914	**0.757**	4.900	2.184
25.000 INSERT (in una tabella indicizzata)	1.555	**1.402**	8.175	3.197
100 SELECT (senza indice)	**2.494**	2.526	3.629	2.760
100 SELECT (su confronto di stringhe)	**3.362**	3.372	13.409	4.640
5.000 SELECT (con indice)	**1.121**	1.162	4.614	1.270
1.000 UPDATE (senza indice)	**0.637**	0.638	1.739	8.410
25.000 UPDATE (con indice)	3.520	**3.104**	18.797	8.134
25.000 UPDATE (con indice su campo di tipo TEXT)	2.408	**1.725**	48.133	6.982
DELETE (senza indice)	4.004	**0.560**	1.509	0.975
DELETE (con indice)	2.068	**0.752**	1.316	2.262
DROP TABLE	0.939	0.254	0.135	**0.015**

Figura 6.9: Tabella delle performance tra i vari DBMS

Come si è potuto constatare dai risultati dei test un applicativo che richiede frequenti operazioni di INSERT e SELECT su un numero di campi e di tabelle limitati, rende SQLite il sistema più performante fra tutti. Questo è il caso della nostra applicazione dove vengono usate molte SELECT e INSERT per gestire il monitoring in tempo reale dei parametri rilevati dai sensori di WSN (nel nostro caso specifico una ogni secondo per ogni sensore collegato).

Per quanto riguarda le dimensioni sul disco 1000 tuple contenenti i dati relativi ad una rilevazione effettuata da un sensore occupano 48,0kb di spazio fisico sul disco. Un rapido calcolo porta a dire che in un giorno di acquisizioni i parametri di un singolo sensore vengono ad occupare poco più di 4Mb di spazio sul disco i quali puoi verrano archiviati in una singola tupla all'interno della tabella *archivio* e cancellati, limitando così lo spazio che si verrebbe ad accumulare con il trascorrere dei giorni. Questo sistema porta ad una valutazione ottimale dello spazio complessivo occupato sul disco (meno di 30Mb occupati lasciando collegati una media di 7 sensori per 2 anni).

Capitolo 7

Conclusioni e sviluppi futuri

In questo lavoro è stato raggiunto l'obiettivo principale di creare una applicazione Web "user friendly" che riuscisse con un alto livello di astrazione a gestire i parametri ambientali di una Wireless Sensor Network. L'eterogeneità delle diverse architetture hardware per le WSN e dei differenti protocolli e topologie di rete utilizzate ha reso necessario lo sviluppo di un sistema in grado di raccogliere ed aggregare questi dati rendendoli disponibili all'utente tramite un comune sistema di interazione.

Questo sistema è stato realizzato con un alto fattore di portabilità (utilizzando ambienti di lavoro diffusi e *open source* quali C++, html, PHP) ed è stato progettato cercando di raggiungere gli obiettivi di aggregazione ed eterogeneità delle WSN. Infatti il sistema centrale e il suo modello di interrogazione dei sensori basato su query permette l'utilizzo di un "linguaggio" comune, indipendente dalle piattaforme hardware sottostanti. L'implementazione fisica delle WSN connesse conseguentemente non modifica la struttura del sistema centrale e del sistema di query. Si è riuscito anche a limitare a poche le modifiche sul software standard del dispositivo della Texas Instruments rendendo l'applicazione Web compatibile e integrabile con altre applicazioni esterne che interagisco con le WSN (il software ED standard non è stato modificato).

Il software e la rete WSN realizzata in questo lavoro, risulta limitata al recupero e alla gestione dell'unico dato fornito dal sensore del dispositivo, ovvero la temperatura. Il sistema rimane comunque valido per ogni tipologia di dato numerico, adeguando lo schema relazionale delle tabelle, con i campi e vincoli

relativi ai dati forniti dalla rete. Le modalità di interrogazione, di segnalazione e comunicazione col sistema centrale non necessitano di sostanziali modifiche.

Per quanto riguarda gli sviluppi futuri, si può notare come essi siano innumerevoli. Per prima cosa si potrebbero aggiungere periferiche e sensori sugli end devices, implementando ulteriori funzionalità sensoristiche come la capacità di rilevare pressione, vibrazioni ed altro ancora, oppure capacità attuative.

Si potrebbe poi approfondire maggiormente la parte relativa alle reti multihop e alle reti formate da più end devices in cascata, arrivando ad utilizzare reti di tipo mesh dove i profili di dinamicità, estensione ed autonomia del sistema sono migliori rispetto alla tipologia a stella utilizzata in questo progetto. Il protocollo di rete (gestione del routing) e l'hardware (poche risorse in termini di memoria e capacità computazionale) utilizzati non permettono facilmente di implementare questa tipologia di rete. Potrebbe risultare un'alternativa, lo sviluppo di un sistema di routing gestito in parte dal nodo e in parte dal software wrapper. La comunicazione dei nodi verso il wrapper avverrebbe in modo automatico, in quanto ogni nodo è a conoscenza del nodo a lui connesso facente parte del "percorso" determinato verso il centro stella (potenzialmente il messaggio può attraversare più nodi). Invece la comunicazione dal centro stella verso i nodi (oppure da nodo a nodo, passando però dal centro stella) verrebbe elaborata dal wrapper, che conoscendo la struttura della rete, calcola il percorso del messaggio, includendolo al suo interno (*payload* oppure *header* di un apposito layer). Ogni nodo che riceve il messaggio da inoltrare, legge l'indirizzo successivo e invia il messaggio al nodo specificato, fino al raggiungimento della destinazione.

Parte III

Appendici

Capitolo 8

Codice

Viene mostrato ora il codice completo relativo ad ogni software scritto e realizzato per rendere operativo l'intero progetto.

8.1 Software wrapperdb.sqlite

```
/*
 * wrapperdp.cpp
 *
 *  Created on:          10-ago-2009
 *      Author:          Andrea Izzo
 *     License:      Public Domain
 */

#include <time.h>
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "windows.h"
#include <sstream>
#include "SqliteLib\SqliteConnection.h"
```

```
#define MESSAGE_LENGHT 22
#define porta "COM1"


string prepareQuery(string val1, string val2, string val3,
                    string val4);
char **explode(char *string, char *separator);
string outSqliteAVG(string temp, string batt, string signal);
void aggiornaArchivio();

using namespace SqliteClass;
using namespace std;

//Variabili Globali

string iquery;
DWORD bytesread;
BYTE charRead[1];
byte read[MESSAGE_LENGHT];
HANDLE hComm;
char **strarray;
char *strdati;

// Creazione puntatore all'oggetto della classe per la
// connessione del db

SqliteConnection *aConn;


/*
 * La funzione printexception serve per stampare a
 * video eventuali errori che si possono
 * verificare durante la lettura dei dati dal database.
 */
```

```
void printexception(SqliteException ex) {

cout << ("\nAN ERROR HAS OCCCOURS....") << "\n" ;
cout << ("STATEMENT : ") << ex.GetStatement() << "\n" ;
cout << ("ERROR NUMBER : ") << ex.GetErrNumber() << "\n";
cout << ("ERROR DESCRIPTION : ") << ex.GetErrDescription()
     << "\n\n\n" ;
}


/*
 * Ciclo Main
 */

int main (int argc, char* argv[]){

aConn = new SqliteConnection();
long countD,countT;
time_t *tmppointer=NULL;
countD = time(tmppointer);

// Creazione del handle relativo all apertura della
// porta in lettura(COM)

hComm = CreateFile(porta,
          GENERIC_READ,
          0, //(share) 0:cannot share the COM port
          0, //security (None)
          OPEN_EXISTING, // creation : open_existing
          0, // we want overlapped operation
          0 // no templates file for COM port...
         );

cout << ("indirizzo porta COM: ")<<hComm<<endl;
```

```
if (hComm == INVALID_HANDLE_VALUE) {
        cout<<("Errore apertura COM");
        exit(0);}// Errore nell'aprire la porta

cout << ("COM Aperta\n\n");

// Apro il database Sqlite, se non presente lo crea

aConn->Open("C:/Program Files/Apache Software
 Foundation/Apache2.2/htdocs/dbsensori.sqlite");

while (true) {

   // sincronizzo la lettura con il primo carattere
   // della stringa ricevuta

   while (*charRead!='$')
        ReadFile(hComm, //porta HANDLE
        charRead, //puntatore al dato da leggere
        1, // numero di bytes da leggere
        &bytesread, // puntatore n# di bytes letti
        NULL //deve essere NULL per Windows CE
   );

   for (int i=0; i<MESSAGE_LENGHT; i++) {

     // si suppone che i dati una volta avviato il
     // trasferimento, arrivino prima di ogni TimeOut

     ReadFile(hComm, charRead, 1, &bytesread, NULL );
     read[i] = (char)(*charRead);
    }
    strdati = (char*)read;
    strarray=explode(strdati, ",");
    iquery = prepareQuery(strarray[0],strarray[1],
```

```
                        strarray[2],strarray[3]);

    // libero la memoria alloca precedentemente

    for (int z = 0; strarray[z]; z++) {
        free(strarray[z]);
    }
    free(strarray);

    try {

     // BeginTrans effettua i cambiamenti
     // direttamente su file

     aConn->BeginTrans();

     // Execute permette di usare comandi sql

     aConn->Execute(iquery);

     // chiude la transizione

     aConn->CommitTrans();
     countT = time(tmppointer);

     // aggiorno la tabella archivio una
     // volta la giorno (86400 sec)

     if (countT>=(countD+86400)) {
        aggiornaArchivio();
        countD = time(tmppointer);
     }
    }
    catch(exception e) {
          cout << "si e' verificato un errore di
```

```
                scrittura sul database";
        aConn->Close();
        delete aConn;
        system("PAUSE");
        return 0;
    }
  }
 }


/*
 * La funzione aggiornaArchivio serve per inserire una
 * tupla contenente la media dei valori prelevati
 * da un sensore nell'arco di una giornata. Una volta
 * aggiornato l'archivio vengono rimosse dal database
 * dei sensori tutte le tuple utilizzate nel calcolo
 * della media.
 */

void aggiornaArchivio() {

cout<<("ARCHIVIO")<<endl;
aConn->Open("C:/Program Files/Apache Software
 Foundation/Apache2.2/htdocs/dbsensori.sqlite");

SqliteRecordSet *aRecordSet;
aRecordSet = new SqliteRecordSet();
string tempAVG,battAVG,strenghtAVG;
string aquery("Select ROUND(AVG(AVG(temperatura),1),
  ROUND(AVG(batteria)), ROUND(AVG(segnale), 1) from sens");


// BeginTrans effettua i cambiamenti direttamente su file

aConn->BeginTrans();
```

```
try {
  // Execute(string,SqliteRecord), RecordSet e' l'oggetto che
  // andra' a contenere i record ricavati

  aConn->Execute(aquery,aRecordSet);
}
catch (SqliteException ex) {
  printexception(ex);
}
if (!aRecordSet->IsEmpty()) {
  // MoveFirst ci fa posizionare al primo record prelevato

  aRecordSet->MoveFirst();

  // Il ciclo avviene finche' non arriviamo all'ultimo record
  while (!aRecordSet->EndOfRecordSet) {
    try {
      // FildValue(n) preleva il valore contenuto nella colonna

        tempAVG = aRecordSet->FieldValue(0);
        battAVG = aRecordSet->FieldValue(1);
        strenghtAVG = aRecordSet->FieldValue(2);

        iquery=outSqliteAVG(tempAVG,battAVG,strenghtAVG);
        aConn->Execute(iquery);
    }
    catch (SqliteException ex) { printexception(ex); }

    // MoveNext ci sposta al prossimo record
    // aRecordSet->MoveNext();
  }
}
else {
        cout << "Non e' stato trovato nessun record!!\n";
}
```

```
aConn->Execute("DELETE FROM sens");
aConn->CommitTrans(); // chiude la transizione
}


/*
 * La funzione outSqliteAVG restituisce una stringa
 * contenente la query che andra' successivamente
 * eseguita per effettuare gli inserimenti all'interno
 * dell'archivio dei sensori.
 */

string outSqliteAVG(string temp, string batt, string signal){

string queryAVG("INSERT INTO archivio
  (temperatura,segnale,batteria,data) VALUES ('");

char dateStr [9];
_strdate( dateStr);

queryAVG.append(temp);
queryAVG.append("','");
queryAVG.append(signal);
queryAVG.append("','");
queryAVG.append(batt);
queryAVG.append("','");
queryAVG.append(dateStr);
queryAVG.append("')");
cout << queryAVG << endl;

return queryAVG;
}

/*
 * La funzione explode restituisce una matrice di
```

```
 * stringhe, ciascuna delle quali e' una parte
 * di string ottenuta dividendo la stringa originale
 * utilizzando separator come separatore di stringa.
 * I parametri della funzione sono, nell'ordine,
 * la stringa da suddividere e' il carattere
 * di separazione.
 */

char **explode(char *string, char *separator) {

char **ret;
int i = 0;

ret = (char **)malloc(sizeof(*ret));
ret[i++] = strtok(string, separator);

while (ret[(i-1)]) {
  ret = (char **)realloc(ret, sizeof(*ret) * (i+1));
  ret[i++] = strtok(NULL, separator);
}

return ret;
}

/*
 * La funzione prepareQuery restituisce una stringa
 * contenente la query che andra' successivamente
 * eseguita per effettuare gli inserimenti all'interno
 * del database.
 */

string prepareQuery(string val1, string val2, string val3,
                    string val4){

long count;
```

```
time_t *tmppointer=NULL;
count = time(tmppointer);
char scount[11];
ltoa(count, scount,10);

scount[10]='\0';
string query("INSERT INTO sens(address,temperatura,
  batteria, segnale,data) VALUES ('");

query.append(val1);
query.append("','");
query.append(val2);
query.append("','");
query.append(val3);
query.append("','");
query.append(val4);
query.append("','");
query.append(scount);
query.append("')");
cout << query << endl;
return query;
}
```

8.2 Software AP e ED

8.2.1 Codice AP modificato

```
//*****************************************************
//   eZ430-RF2500 Temperature Sensor Access Point
//
//   Description: This is the Access Point software for
//                the eZ430-2500RF
//                Temperature Sensing demo
//
//
```

```
//   L. Westlund
//   Version    1.02
//   Texas Instruments, Inc
//   November 2007
//   Built with IAR Embedded Workbench Version: 4.09A
//*************************************************
//Change Log:
//*************************************************
//Version:  1.02
//Comments: Changed Port toggling to abstract method
//          Removed ToggleLED
//          Fixed comment typos/errors
//          Changed startup string to 1.02
//Version:  1.01
//Comments: Added support for SimpliciTI 1.0.3
//          Changed RSSI read method
//          Added 3 digit temperature output for 100+F
//          Changed startup string to 1.01
//Version:  1.00
//Comments: Initial Release Version
//*************************************************

#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"

#include "msp430x22x4.h"
#include "vlo_rand.h"

#define MESSAGE_LENGTH 3
void TXString( char* string, int length );
void MCU_Init(void);
```

```
void transmitData(int addr, signed char rssi,  char msg[
   MESSAGE_LENGTH] );
void transmitDataString(char addr[4],char rssi[3], char msg
   [MESSAGE_LENGTH]);
void createRandomAddress();


__no_init volatile int tempOffset @ 0x10F4; // Temperature
   offset set at production
__no_init volatile char Flash_Addr[4] @ 0x10F0; // Flash
   address set randomly

// reserve space for the maximum possible peer Link IDs
static linkID_t sLID[NUM_CONNECTIONS];
static uint8_t  sNumCurrentPeers;

// callback handler
static uint8_t sCB(linkID_t);

// work loop semaphores
static uint8_t sPeerFrameSem;
static uint8_t sJoinSem;
static uint8_t sSelfMeasureSem;

// mode data verbose = default, deg F = default
char verboseMode = 0; // Modifica: disattivato la modalita'
    estesa
char degCMode = 1; // Modifica: abilitata la temperatura in
    gradi celsius

void main (void)
{
  addr_t lAddr;
  bspIState_t intState;

  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  {
```

```
// delay loop to ensure proper startup before SimpliciTI
   increases DCO
// This is typically tailored to the power supply used,
   and in this case
// is overkill for safety due to wide distribution.
  volatile int i;
  for(i = 0; i < 0xFFFF; i++){}
}
if( CALBC1_8MHZ == 0xFF )                  // Do not run
   if cal values are erased
{
  volatile int i;
  P1DIR |= 0x03;
  BSP_TURN_ON_LED1();
  BSP_TURN_OFF_LED2();
  while(1)
  {
    for(i = 0; i < 0x5FFF; i++){}
    BSP_TOGGLE_LED2();
    BSP_TOGGLE_LED1();
  }
}

BSP_Init();

if( Flash_Addr[0] == 0xFF &&
    Flash_Addr[1] == 0xFF &&
    Flash_Addr[2] == 0xFF &&
    Flash_Addr[3] == 0xFF )
{
  createRandomAddress();                   // set Random
     device address at initial startup
}
lAddr.addr[0]=Flash_Addr[0];
lAddr.addr[1]=Flash_Addr[1];
lAddr.addr[2]=Flash_Addr[2];
lAddr.addr[3]=Flash_Addr[3];
```

```
SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);

MCU_Init();
//Transmit splash screen and network init notification
TXString( (char*)splash, sizeof splash);
TXString( "\r\nInitializing Network....", 26 );

SMPL_Init(sCB);

// network initialized
TXString( "Done\r\n", 6);

// main work loop
while (1)
{
  // Wait for the Join semaphore to be set by the receipt
      of a Join frame from a
  // device that supports and End Device.

  if (sJoinSem && (sNumCurrentPeers < NUM_CONNECTIONS))
  {
    // listen for a new connection
    SMPL_LinkListen(&sLID[sNumCurrentPeers]);
    sNumCurrentPeers++;
    BSP_ENTER_CRITICAL_SECTION(intState);
    if (sJoinSem)
    {
      sJoinSem--;
    }
    BSP_EXIT_CRITICAL_SECTION(intState);
  }

  // if it is time to measure our own temperature...
  if(sSelfMeasureSem)
  {
    char msg [6];
    char addr[] = {"HUB0"};
```

```
char rssi[] = {"000"};
int degC, volt;
volatile long temp;
int results[2];

ADC10CTL1 = INCH_10 + ADC10DIV_4;     // Temp Sensor
   ADC10CLK/5
ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON +
   ADC10IE + ADC10SR;
for( degC = 240; degC > 0; degC-- );  // delay to
   allow reference to settle
ADC10CTL0 |= ENC + ADC10SC;           // Sampling and
    conversion start
__bis_SR_register(CPUOFF + GIE);      // LPM0 with
   interrupts enabled
results[0] = ADC10MEM;

ADC10CTL0 &= ~ENC;

ADC10CTL1 = INCH_11;                  // AVcc/2
ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON +
   ADC10IE + REF2_5V;
for( degC = 240; degC > 0; degC-- );  // delay to
   allow reference to settle
ADC10CTL0 |= ENC + ADC10SC;           // Sampling and
    conversion start
__bis_SR_register(CPUOFF + GIE);      // LPM0 with
   interrupts enabled
results[1] = ADC10MEM;
ADC10CTL0 &= ~ENC;
ADC10CTL0 &= ~(REFON + ADC10ON);      // turn off A/D
    to save power

// oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10
   *423/1024 - 278
// the temperature is transmitted as an integer where
    32.1 = 321
```

```
    // hence 4230 instead of 423
    temp = results[0];
    degC = (((temp - 673) * 4230) / 1024);
    if( tempOffset != 0xFFFF )
    {
      degC += tempOffset;
    }

    temp = results[1];
    volt = (temp*25)/512;

    msg[0] = degC&0xFF;
    msg[1] = (degC>>8)&0xFF;
    msg[2] = volt;
    transmitDataString(addr, rssi, msg );
    BSP_TOGGLE_LED1();
    sSelfMeasureSem = 0;
  }

  // Have we received a frame on one of the ED
     connections?
  // No critical section -- it doesn't really matter much
      if we miss a poll
  if (sPeerFrameSem)
  {
    uint8_t     msg[MAX_APP_PAYLOAD], len, i;

    // process all frames waiting
    for (i=0; i<sNumCurrentPeers; ++i)
    {
      if (SMPL_Receive(sLID[i], msg, &len) ==
         SMPL_SUCCESS)
      {
        ioctlRadioSiginfo_t sigInfo;
        sigInfo.lid = sLID[i];
        SMPL_Ioctl(IOCTL_OBJ_RADIO,
           IOCTL_ACT_RADIO_SIGINFO, (void *)&sigInfo);
```

```
          transmitData( i, (signed char)sigInfo.sigInfo[0],
              (char*)msg );
          BSP_TOGGLE_LED2();
          BSP_ENTER_CRITICAL_SECTION(intState);
          sPeerFrameSem--;
          BSP_EXIT_CRITICAL_SECTION(intState);
        }
      }
    }
  }
}

/*-------------------------------------------
*
-------------------------------------------*/
void createRandomAddress()
{
  unsigned int rand, rand2;
  do
  {
    rand = TI_getRandomIntegerFromVLO();    // first byte
       can not be 0x00 of 0xFF
  }
  while( (rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000
      );
  rand2 = TI_getRandomIntegerFromVLO();

  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO to 1
     MHz
  DCOCTL = CALDCO_1MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN1;             // MCLK/3 for
     Flash Timing Generator
  FCTL3 = FWKEY + LOCKA;                    // Clear LOCK &
      LOCKA bits
  FCTL1 = FWKEY + WRT;                      // Set WRT bit
     for write operation
```

```
  Flash_Addr[0]=(rand>>8) & 0xFF;
  Flash_Addr[1]=rand & 0xFF;
  Flash_Addr[2]=(rand2>>8) & 0xFF;
  Flash_Addr[3]=rand2 & 0xFF;

  FCTL1 = FWKEY;                              // Clear WRT
     bit
  FCTL3 = FWKEY + LOCKA + LOCK;               // Set LOCK &
     LOCKA bit
}

/*-------------------------------------------
*
-------------------------------------------*/
void transmitData(int addr, signed char rssi,  char msg[
   MESSAGE_LENGTH] )
{
  char addrString[4];
  char rssiString[3];
  volatile signed int rssi_int;

  addrString[0] = '0';
  addrString[1] = '0';
  addrString[2] = '0'+(((addr+1)/10)%10);
  addrString[3] = '0'+((addr+1)%10);
  rssi_int = (signed int) rssi;
  rssi_int = rssi_int+128;
  rssi_int = (rssi_int*100)/256;
  rssiString[0] = '0'+(rssi_int%10);
  rssiString[1] = '0'+((rssi_int/10)%10);
  rssiString[2] = '0'+((rssi_int/100)%10);

  transmitDataString( addrString, rssiString, msg );
}

/*-------------------------------------------
*
```

```
-----------------------------------------------*/
void transmitDataString(char addr[4],char rssi[3], char msg
   [MESSAGE_LENGTH] )
{
  char temp_string[] = {" XX.XC"};
  int temp = msg[0] + (msg[1]<<8);

  if( !degCMode )
  {
    temp = (((float)temp)*1.8)+320;
    temp_string[5] = 'F';
  }
  if( temp < 0 )
  {
    temp_string[0] = '-';
    temp = temp * -1;
  }
  else if( ((temp/1000)%10) != 0 )
  {
    temp_string[0] = '0'+((temp/1000)%10);
  }
  temp_string[4] = '0'+(temp%10);
  temp_string[2] = '0'+((temp/10)%10);
  temp_string[1] = '0'+((temp/100)%10);

  if( verboseMode )
  {
    char output_verbose[] = {"\r\nNode:XXXX,Temp:-XX.XC,
       Battery:X.XV,Strength:XXX%,RE:no "};

    output_verbose[46] = rssi[2];
    output_verbose[47] = rssi[1];
    output_verbose[48] = rssi[0];

    output_verbose[17] = temp_string[0];
    output_verbose[18] = temp_string[1];
    output_verbose[19] = temp_string[2];
```

```
    output_verbose[20] = temp_string[3];
    output_verbose[21] = temp_string[4];
    output_verbose[22] = temp_string[5];

    output_verbose[32] = '0'+(msg[2]/10)%10;
    output_verbose[34] = '0'+(msg[2]%10);
    output_verbose[7] = addr[0];
    output_verbose[8] = addr[1];
    output_verbose[9] = addr[2];
    output_verbose[10] = addr[3];
    TXString(output_verbose, sizeof output_verbose );
  }
  else
  {
        /*
         * Modifica: eliminati dal output_short i caratteri
             speciali /r/n e
         * rimappato gli indirizzi all'interno dell'
            output_short
         */
    char output_short[] = {"$ADDR,-XX.XC,V.C,RSI,N#"};

    output_short[17] = rssi[2];
    output_short[18] = rssi[1];
    output_short[19] = rssi[0];

    output_short[6] = temp_string[0];
    output_short[7] = temp_string[1];
    output_short[8] = temp_string[2];
    output_short[9] = temp_string[3];
    output_short[10] = temp_string[4];
    output_short[11] = temp_string[5];

    output_short[13] = '0'+(msg[2]/10)%10;
    output_short[15] = '0'+(msg[2]%10);
    output_short[1] = addr[0];
    output_short[2] = addr[1];
```

```
    output_short[3] = addr[2];
    output_short[5] = addr[3];
    TXString(output_short, sizeof output_short );
  }
}

/*-------------------------------------------
*
-------------------------------------------*/
void TXString( char* string, int length )
{
  int pointer;
  for( pointer = 0; pointer < length; pointer++)
  {
    volatile int i;
    UCA0TXBUF = string[pointer];
    while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX
       buffer ready?
  }
}

/*-------------------------------------------
*
-------------------------------------------*/
void MCU_Init()
{
  BCSCTL1 = CALBC1_8MHZ;                    // Set DCO
  DCOCTL = CALDCO_8MHZ;

  BCSCTL3 |= LFXT1S_2;                      // LFXT1 = VLO
  TACCTL0 = CCIE;                           // TACCR0
     interrupt enabled
  TACCR0 = 12000;                           // ~1 second
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

  P3SEL |= 0x30;                            // P3.4,5 =
     USCI_A0 TXD/RXD
```

```
  UCA0CTL1 = UCSSEL_2;                      // SMCLK
  UCA0BR0 = 0x41;                           // 9600 from 8
     Mhz
  UCA0BR1 = 0x3;
  UCA0MCTL = UCBRS_2;
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize
      USCI state machine**
  IE2 |= UCA0RXIE;                          // Enable
     USCI_A0 RX interrupt
  __enable_interrupt();
}
/*-------------------------------------------------
* Runs in ISR context. Reading the frame should be done in
   the
* application thread not in the ISR thread.
-------------------------------------------------*/
static uint8_t sCB(linkID_t lid)
{
  if (lid)
  {
    sPeerFrameSem++;
  }
  else
  {
    sJoinSem++;
  }
  // leave frame to be read by application.
  return 0;
}

/*-------------------------------------------------
* ADC10 interrupt service routine
-------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
```

```
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF
      bit from 0(SR)
}

/*-------------------------------------------------
* Timer A0 interrupt service routine
-------------------------------------------------*/
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  sSelfMeasureSem = 1;
}

/*-------------------------------------------------
* USCIA interrupt service routine
-------------------------------------------------*/
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  char rx = UCA0RXBUF;
  if ( rx == 'V' || rx == 'v' )
  {
    verboseMode = 1;
  }
  else if ( rx == 'M' || rx == 'm' )
  {
    verboseMode = 0;
  }
  else if ( rx == 'F' || rx == 'f' )
  {
    degCMode = 0;
  }
  else if ( rx == 'C' || rx == 'c' )
  {
    degCMode = 1;
  }
}
```

8.2.2 Codice ED standard

```
//*****************************************************
//   eZ430-RF2500 Temperature Sensor End Device
//
//   Description: This is the End Device software for the
   eZ430-2500RF
//                Temperature Sensing demo
//
//
//   L. Westlund
//   Version    1.02
//   Texas Instruments, Inc
//   November 2007
//   Built with IAR Embedded Workbench Version: 4.09A
//*****************************************************
//Change Log:
//*****************************************************
//Version:  1.02
//Comments: Changed Port toggling to abstract method
//          Fixed comment typos
//Version:  1.01
//Comments: Added support for SimpliciTI 1.0.3
//          Added Flash storage/check of Random address
//          Moved LED toggle to HAL
//Version:  1.00
//Comments: Initial Release Version
//*****************************************************

#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "vlo_rand.h"
```

```
void linkTo(void);
void MCU_Init(void);

__no_init volatile int tempOffset @ 0x10F4; // Temperature
   offset set at production
__no_init volatile char Flash_Addr[4] @ 0x10F0; // Flash
   address set randomly

void createRandomAddress();

void main (void)
{
  addr_t lAddr;
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  {
  // delay loop to ensure proper startup before SimpliciTI
     increases DCO
  // This is typically tailored to the power supply used,
     and in this case
  // is overkill for safety due to wide distribution.
    volatile int i;
    for(i = 0; i < 0xFFFF; i++){}
  }
  if( CALBC1_8MHZ == 0xFF )                 // Do not run
     if cal values are erased
  {
    volatile int i;
    P1DIR |= 0x03;
    BSP_TURN_ON_LED1();
    BSP_TURN_OFF_LED2();
    while(1)
    {
      for(i = 0; i < 0x5FFF; i++){}
      BSP_TOGGLE_LED2();
      BSP_TOGGLE_LED1();
    }
  }
```

```
// SimpliciTI will change port pin settings as well
P1DIR = 0xFF;
P1OUT = 0x00;
P2DIR = 0x27;
P2OUT = 0x00;
P3DIR = 0xC0;
P3OUT = 0x00;
P4DIR = 0xFF;
P4OUT = 0x00;

BSP_Init();

if( Flash_Addr[0] == 0xFF &&
    Flash_Addr[1] == 0xFF &&
    Flash_Addr[2] == 0xFF &&
    Flash_Addr[3] == 0xFF )
{
  createRandomAddress();                    // set Random
     device address at initial startup
}
lAddr.addr[0]=Flash_Addr[0];
lAddr.addr[1]=Flash_Addr[1];
lAddr.addr[2]=Flash_Addr[2];
lAddr.addr[3]=Flash_Addr[3];
SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);
BCSCTL1 = CALBC1_8MHZ;                      // Set DCO
   after random function
DCOCTL = CALDCO_8MHZ;

BCSCTL3 |= LFXT1S_2;                        // LFXT1 = VLO
TACCTL0 = CCIE;                             // TACCR0
   interrupt enabled
TACCR0 = 12000;                             // ~ 1 sec
TACTL = TASSEL_1 + MC_1;                    // ACLK, upmode
```

```
  // keep trying to join until successful. toggle LEDS to
     indicate that
  // joining has not occurred. LED3 is red but labeled LED
     4 on the EXP
  // board silkscreen. LED1 is green.
  while (SMPL_NO_JOIN == SMPL_Init((uint8_t (*)(linkID_t))
     0))
  {
    BSP_TOGGLE_LED1();
    BSP_TOGGLE_LED2();;
    __bis_SR_register(LPM3_bits + GIE);     // LPM3 with
       interrupts enabled
  }
  // unconditional link to AP which is listening due to
     successful join.
  linkTo();
}

void createRandomAddress()
{
  unsigned int rand, rand2;
  do
  {
    rand = TI_getRandomIntegerFromVLO();    // first byte
       can not be 0x00 of 0xFF
  }
  while( (rand & 0xFF00)==0xFF00 || (rand & 0xFF00)==0x0000
      );
  rand2 = TI_getRandomIntegerFromVLO();

  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO to 1
     MHz
  DCOCTL = CALDCO_1MHZ;
  FCTL2 = FWKEY + FSSEL0 + FN1;             // MCLK/3 for
     Flash Timing Generator
  FCTL3 = FWKEY + LOCKA;                    // Clear LOCK &
      LOCKA bits
```

```
  FCTL1 = FWKEY + WRT;                          // Set WRT bit
     for write operation

  Flash_Addr[0]=(rand>>8) & 0xFF;
  Flash_Addr[1]=rand & 0xFF;
  Flash_Addr[2]=(rand2>>8) & 0xFF;
  Flash_Addr[3]=rand2 & 0xFF;

  FCTL1 = FWKEY;                                // Clear WRT
     bit
  FCTL3 = FWKEY + LOCKA + LOCK;                 // Set LOCK &
     LOCKA bit
}

void linkTo()
{
  linkID_t linkID1;
  uint8_t  msg[3];

  // keep trying to link...
  while (SMPL_SUCCESS != SMPL_Link(&linkID1))
  {
    __bis_SR_register(LPM3_bits + GIE);     // LPM3 with
       interrupts enabled
    BSP_TOGGLE_LED1();
    BSP_TOGGLE_LED2();
  }

  // Turn off all LEDs
  if (BSP_LED1_IS_ON())
  {
    BSP_TOGGLE_LED1();
  }
  if (BSP_LED2_IS_ON())
  {
    BSP_TOGGLE_LED2();
  }
```

```
while (1)
{
  volatile long temp;
  int degC, volt;
  int results[2];
  SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, ""
     );
  __bis_SR_register(LPM3_bits+GIE);       // LPM3 with
     interrupts enabled
  SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, ""
     );

  BSP_TOGGLE_LED2();
  ADC10CTL1 = INCH_10 + ADC10DIV_4;       // Temp Sensor
     ADC10CLK/5
  ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON +
     ADC10IE + ADC10SR;
  for( degC = 240; degC > 0; degC-- );    // delay to
     allow reference to settle
  ADC10CTL0 |= ENC + ADC10SC;             // Sampling and
      conversion start
  __bis_SR_register(CPUOFF + GIE);        // LPM0 with
     interrupts enabled
  results[0] = ADC10MEM;

  ADC10CTL0 &= ~ENC;

  ADC10CTL1 = INCH_11;                     // AVcc/2
  ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON +
     ADC10IE + REF2_5V;
  for( degC = 240; degC > 0; degC-- );    // delay to
     allow reference to settle
  ADC10CTL0 |= ENC + ADC10SC;             // Sampling and
      conversion start
  __bis_SR_register(CPUOFF + GIE);        // LPM0 with
     interrupts enabled
  results[1] = ADC10MEM;
```

```
ADC10CTL0 &= ~ENC;
ADC10CTL0 &= ~(REFON + ADC10ON);        // turn off A/D
    to save power

// oC = ((A10/1024)*1500mV)-986mV)*1/3.55mV = A10
   *423/1024 - 278
// the temperature is transmitted as an integer where
   32.1 = 321
// hence 4230 instead of 423
temp = results[0];
degC = ((temp - 673) * 4230) / 1024;
if( tempOffset != 0xFFFF )
{
  degC += tempOffset;
}
/*message format,  UB = upper Byte, LB = lower Byte
--------------------------------
|degC LB | degC UB |  volt LB |
--------------------------------
   0         1          2
*/

temp = results[1];
volt = (temp*25)/512;
msg[0] = degC&0xFF;
msg[1] = (degC>>8)&0xFF;
msg[2] = volt;

if (SMPL_SUCCESS == SMPL_Send(linkID1, msg, sizeof(msg)
   ))
{
  BSP_TOGGLE_LED2();
}
else
{
  BSP_TOGGLE_LED2();
  BSP_TOGGLE_LED1();
```

```
    }
  }
}

/*-------------------------------------------------
* ADC10 interrupt service routine
-------------------------------------------------*/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF
      bit from 0(SR)
}

/*-------------------------------------------------
* Timer A0 interrupt service routine
-------------------------------------------------*/
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  __bic_SR_register_on_exit(LPM3_bits);        // Clear
     LPM3 bit from 0(SR)
}
```

8.3 Applicazione Web

8.3.1 SensiNet.php

```
<HTML>
 <HEAD>
 <link rel="stylesheet" href="stilehome.css" type="text/css
    " />
 <img class="logo" alt="logo_SensiNet" src="logoSensiNet.
    gif">
 <TITLE>SensiNet</TITLE>
</HEAD>
<body>
 <script language="JavaScript">
  function doRefresh() {
  document.location.reload();}
  window.setTimeout("doRefresh();", 1000);
 </script>
 <ul class="menu">
  <li><a href="login.php">Login</a></li>
  <li><a href="SensiNet_rep.php">Tabella reporting</a></li>
 </ul>

 <!-- Tabella contenente gli ultimi rilevamenti dei sensori
    -->

 <div class="time"><?php echo date("d-m-Y H:i:s");?></div>
 <div align="center">
  <table>
   <thead><tr> <th>Sensore</th> <th>Temperatura</th> <th>
      Batteria
   </th> <th>Segnale</th> </tr></thead>

  <?php

   /* Connette al database ODBC invocando i driver */

   $dsn = 'sqlite:C:\Program Files\Apache Software
```

```
    Foundation\Apache2.2\htdocs\dbsensori.sqlite';

 try {
  $dbh = new PDO($dsn);
 } catch (PDOException $e) {
  echo 'Connection failed: ' . $e->getMessage();
 }
 $dbh->beginTransaction();

 /* Recupera i dati dei sensori dal database */

 $query="select *, Max(data) from sens";
 $prep_query=$dbh->query($query);
 $arr = $prep_query->fetchAll();
 $i = 0;
 foreach ($arr as $result){
  $i++;
  if ($i%2) {$riga = "dispari";}
 else {$riga = "pari";}
 ?>
 <tbody>
  <tr class="<?=$riga?>">
   <td>Sensore <?= $result['address']?></td>
   <td><?= $result['temperatura']?> </td>
   <td><?= $result['batteria']?>V</div></td>
   <td><?php
      echo '
      <div class="graphcont"><div class="graph">
      <div class="bar" style="width:'.$result['segnale'].
      '%;">'.$result['segnale'].'%
        </div></div></div> ';?></td></tr>

      <?php  }/**/?>
   </tbody>
  </table>
 </div>
</body>
```

```
</HTML>
```

8.3.2 SensiNet_rep.php

```
<HTML>
 <HEAD>
 <link rel="stylesheet" href="stilehome.css" type="text/css
    " />
 <img class="logo" alt="logo_SensiNet" src="logoSensiNet.
    gif">
 <TITLE>SensiNet</TITLE>
 </HEAD>
 <body>
  <?php $type= $_GET['tipo']; ?>
  <ul class="menu">

  <?php if ($type=="")
   {?><li><a href="login.php">Login</a></li>
   <?php } ?>

  <?php if ($type=="log")
   {?><li><a href="home.php?tipo=Tesi">Tesi</a></li>
   <?php } ?>

  <?php if ($type=="log")
   {?><li><a href="home.php?tipo=Articoli">Articoli</a></li
      >
   <?php } ?>

  <?php if ($type=="log")
   {?><li><a href="home.php?tipo=Documenti">Documentazioni
      </a></li>
   <?php } ?>

  <?php if ($type=="log")
   {?><li><a href="sensori.php">Sensori</a></li><?php }?>
```

```
<?php if ($type=="")
 {?><li><a href="SensiNet.php">Sensori</a></li><?php }?>

<?php if ($type=="log")
 {?><li><a href="home.php?tipo=logout">Logout</a></li><?
    php } ?>

</ul>
<!-- Tabella contenente l'archivio dei sensori-->
<div class="time"><?php echo date("d-m-Y H:i:s");?></div>
<div align="center">
 <table>
      <thead><tr> <th>Sensore</th> <th>Temperatura</th>
      <th>Batteria</th> <th>Segnale</th> <th>Data</th> </
         tr></thead>

      <?php
       /* Connette al database ODBC invocando i driver */

      $dsn = 'sqlite:C:\Program Files\Apache Software
     Foundation\Apache2.2\htdocs\dbsensori.sqlite';
      try {
       $dbh = new PDO($dsn);
      } catch (PDOException $e) {
       echo 'Connection failed: ' . $e->getMessage();
      }
      $dbh->beginTransaction();

      /* Recupera i dati dei sensori dal database */

      $query="SELECT * FROM archivio";
      $prep_query=$dbh->query($query);
      $arr = $prep_query->fetchAll();
      $i = 0;
      foreach ($arr as $result){
     $i++;
       if ($i%2) {$riga = "dispari";}
```

```
          else {$riga = "pari";}
        ?>

      <tbody>
         <tr class="<?=$riga?>">
          <td>Sensore <?= $result['address']?></td>
          <td><?= $result['temperatura']?> </td>
          <td><?= $result['batteria']?>V</div></td>
          <td><?php
           $result['batteria']=rand(0,100);
           echo '
           <div class="graphcont"><div class="graph">
           <div class="bar" style="width:'.$result['segnale
              '].
          '%;">'.$result['segnale'].'%
           </div></div></div> ';?></td>
          <td><?= $result['data']?></td></tr>
        <?php  }/**/?>
     </tbody>
    </table>
   </div>
  </body>
</HTML>
```

8.3.3 login.php

```
<html>
 <link rel="stylesheet" href="stilehome.css" type="text/css
    " />
 <img class="logo" alt="logo_SensiNet" src="logoSensiNet.
    gif">
 <head>
  <title>Login</title>
 </head>
 <?php
  session_start();
  $idutente = false;
```

```
/* Connette al database ODBC invocando i driver */

$dsn = 'sqlite:C:\Program Files\Apache Software
 Foundation\Apache2.2\htdocs\database.sqlite';
try {
 $dbh = new PDO($dsn);
} catch (PDOException $e) {
 echo 'Connection failed: ' . $e->getMessage();
}
$dbh->beginTransaction();
$loggato = FALSE;

// setto il token solo se non e' gia' settato

if (!isset($_SESSION['token'])){
 $token = md5(uniqid(rand(), true));
 $_SESSION['token'] = $token;
} else {

// tolgo le eventuali info sul vecchio login

$query_a=("SELECT * FROM authlogin WHERE
 (authcode='{$_SESSION['token']}')");
$prep_query=$dbh->query($query_a);
$a = $prep_query->fetchAll();
if ($a[0]['authcode'] == $_SESSION['token'] ){

 // siamo loggati?>

 <div class="infoLogin"> Sei gi  loggato
  ..aspetta il reindirezzamento</div>

 <meta http-equiv="refresh" content="0;
  url=sensori.php?tipo=sens">

 <?php
```

```
 $loggato = TRUE;
 }
}
if(count($_POST)>0) {

 // controllo user e psw

 if (isset($_POST['userid']) && isset($_POST['passwd'])
  && !$loggato) {
    $username = $_POST['userid'];
      $password = md5($_POST['passwd']);
      $token=$_SESSION['token'];
      $query_b=("SELECT * FROM utente WHERE
     (username='{$username}')");
      $prep_query=$dbh->query($query_b);
      $a = $prep_query->fetchAll();
      if (count($a)==0) {?>

       <div class="infoLogin"> l'utente non esiste in
      database</div>

       <meta http-equiv="refresh" content="2;
     url=login.php">

     <?php
     } else {

   //password corretta.

     if ($password == $a[0]['password']) {
      $idutente = $a[0]['idutente'];

      // tolgo le eventuali info sul vecchio login

      $query_a=("DELETE FROM authlogin WHERE
     (idutente= '{$idutente}')");
```

```
    $prep_query=$dbh->exec($query_a);
    $dbh->commit();

    // inserisco la nuova tupla per il login

    $dbh->beginTransaction();
    $query_a=("INSERT INTO authlogin (idutente,
       authcode)
   VALUES ('{$idutente}','{$token}')");
    $prep_query=$dbh->exec($query_a);
    $dbh->commit();
    ?>

    <div class="infoLogin"> Login avvenuta con
       successo!</div>
  <?php
     if ($idutente == 1) {?>
    <meta http-equiv="refresh" content="2; url=Admin.
       php">
   <?php }
     if ($idutente != 1) {?>
    <meta http-equiv="refresh" content="2; url=sensori
       .php?
    tipo=sens"> <?php }

     } else { ?>
        <div class="infoLogin"> Password sbagliata!
      </div>
        <meta http-equiv="refresh" content="2; url=
           login.php">

           <?php
           }
       }
      }
     } else {
      ?>
```

```
           <body>
            <form method=post action="login.php">
             <table cellpadding="4">
        <tr class='dispari'>
                 <td colspan="2" align="left">inserite nome
                     utente
               e password:</td>
        </tr>
        <tr>
        <td>nome utente:</td>
                 <td><input type="text" name="userid"></td
                    ></tr>
         <tr>
                  <td>password:</td>
                  <td><input type="password" name="passwd
                     "></td></tr>
         <tr>
                  <td colspan="2">
                  <input type="submit" name="invio" value="
                     invio">
                    
                  <input type="reset" name="cancella" value
                     ="cancella">
                 </td></tr>
        </table>
       </form>
      </body>
    </html>
<?php } ?>
```

8.3.4 sensori.php

```
<HTML>
 <HEAD>
 <link rel="stylesheet" href="stilehome.css" type="text/css
    " />
```

```
<img class="logo" alt="logo_SensiNet" src="logoSensiNet.
   gif">
<TITLE>SensNet</TITLE>
</HEAD>
<script language="JavaScript">
 function doRefresh() {
  document.location.reload();
 }
 window.setTimeout("doRefresh();", 1000);
</script>

<?php session_start();
  $idutente = id_utente_loggato();
  $type= $_GET['tipo'];
  if ($idutente == FALSE) {
   ?>
   <div class="infologin"> Non hai i permessi per accedere
   al sito. Esegui il Login </div>

   <meta http-equiv="refresh" content="2;
   url=SensiNet.php">
   <?php
  } else {?>
   <body>
    <ul class="menu">
       <li><a href="home.php?tipo=Tesi">Tesi</a></li>
       <li><a href="home.php?tipo=Articoli">Articoli</a></
          li>
       <li><a href="home.php?tipo=Documenti">
          Documentazioni</a></li>
       <li><a href="SensiNet_rep.php?tipo=log">Tabella
          Reporting</a>
     </li>
       <li><a href="home.php?tipo=logout">Logout</a></li>
    </ul>
```

```
<!-- Tabella contenente gli ultimi rilevamenti dei
   sensori -->

<div class="time"><?php echo date("H:i:s d-m-Y ");?></
   div>
<div align="center">
   <table>
    <thead><tr> <th>Sensore</th> <th>Temperatura</th>
   <th>Batteria</th> <th>Segnale</th> </tr></thead>
    <?php

   /* Connette al database ODBC invocando i driver */

     $dsn = 'sqlite:C:\Program Files\Apache Software
    Foundation\Apache2.2\htdocs\dbsensori.sqlite';
     try {
      $dbh = new PDO($dsn);
     } catch (PDOException $e) {
      echo 'Connection failed: ' . $e->getMessage();
     }
     $dbh->beginTransaction();

     /* Recupera i dati dei sensori dal database */

      $query="select *, Max(data) from sens";
      $prep_query=$dbh->query($query);
      $arr = $prep_query->fetchAll();
      $i = 0;
      foreach ($arr as $result){
       $i++;
       if ($i%2) {$riga = "dispari";}
       else {$riga = "pari";}
       ?>
       <tbody>
        <tr class="<?=$riga?>">
         <td>Sensore <?= $result['address']?></td>
       <td><?= $result['temperatura']?> </td>
```

```
                <td><?= $result['batteria']?>V</div></td>
                <td><?php echo '
                    <div class="graphcont"><div class="graph">
               <div class="bar" style="width:'.$result
                ['segnale'].'%;">'.$result['segnale'].'%
                    </div></div></div> ';?></td>
                </tr>
                  <?php  }
               }

/*
 * -------------------------------------------------------
 * Ritorna l'id dell'utente loggato oppure FALSE se
 * non lo e'
 * -------------------------------------------------------
 */

function id_utente_loggato(){

 // tolgo le eventuali info sul vecchio login
 if (!isset($_SESSION['token']))
  return FALSE;
  $query_a=("SELECT * FROM authlogin WHERE
   (authcode='{$_SESSION['token']}')");

  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
   $dbg = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbg->beginTransaction();
  $prep_query=$dbg->query($query_a);
  $a = $prep_query->fetchAll();
  if ($a[0]['authcode'] == $_SESSION['token'] ){
```

```
   // siamo loggati

   return $a[0]['idutente'];
   } else
   return FALSE;
 }


/*
 * ------------------------------------------------
 *  Funzione per effettuare il logout
 * ------------------------------------------------
 */

function logout(){
 unset($_SESSION['token']);
 ?>
 <div class="infologin"> Logout eseguito </div>
 <meta http-equiv="refresh" content="0; url=SensiNet.php">
 <?php } ?>

    </tbody>
   </table>
  </div>
 </body>
</HTML>
```

8.3.5 home.php

```
<html>
 <body>
 <link rel="stylesheet" href="stilehome.css" type="text/css
    " />
 <img class="logo" alt="logo_SensiNet" src="logoSensiNet.
    gif">
 <div class="time"><?php echo date("d-m-Y");?></div>
 <?PHP
```

```
  session_start();
  $idutente = id_utente_loggato();
  if ($idutente == FALSE) {
   ?>
   <div class="infologin"> Non hai i permessi
    per accedere al sito. Esegui il Login </div>

   <meta http-equiv="refresh" content="2; url=SensiNet.php
      ">
   <?php
  }
  if ($idutente == 1) {?>
   <meta http-equiv="refresh" content="2; url=Admin.php">
   <?php
  }
  else {
   ?>
   <ul class="menu">
    <li><a href="home.php?tipo=Tesi">Tesi</a></li>
    <li><a href="home.php?tipo=Articoli">Articoli</a></li>
    <li><a href="home.php?tipo=Documenti">Documentazioni</a
       ></li>
    <li><a href="sensori.php">Sensori</a></li>
    <li><a href="home.php?tipo=logout">Logout</a></li>
  </ul>

<?php

  /* Connette al database ODBC invocando i driver */

  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\dbsensori.sqlite';
  try {
   $dbh = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
```

```
  $dbh->beginTransaction();

  // Recupero i parametri da passare alla funzione
  // predefinita per l'upload

  $type= $_GET['tipo'];?>
  <div class="sezione"><?php echo 'Gestione '.$type;?>
  </div>
  <?php

   if ($type=='Tesi') {
    tabella_tesi();}

   if ($type=='Articoli') {
    tabella_art();}

   if ($type=='Documenti') {
    tabella_doc();}

   if ($type=='logout') logout();
  }



/*
 * ---------------------------------------------
 *  Funzione per la creazione della tabella Tesi
 *  ---------------------------------------------
 */

function tabella_tesi(){ ?>
 <form class="bottone_upload" action="inserimento.php?
    upload=Tesi"
   method="post">
   <input type="submit" value="upload">
 </form>
```

```
<form class="cerca" action="cerca.php?cerca=Tesi" method="
   post">
  <input type="submit" value="cerca">
</form>
<table>
 <thead><tr> <th>Tesi</th> <th>Autore</th> <th>Relatore</
    th>
   <th>Gruppo</th> <th>Data</th></tr></thead>

 <?php

  /* Connette al database ODBC invocando i driver */
  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
   $dbh = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbh->beginTransaction();
  $query_a="SELECT * FROM Tesi";
  $prep_query=$dbh->query($query_a);
  $arr = $prep_query->fetchAll();
  $i = 0;
  foreach ($arr as $result){
       $i++;
       if ($i%2) {$riga = "dispari";}
       else {$riga = "pari";}
  ?>
  <tbody>
   <tr class="<?=$riga?>">
    <td><a href=<?=$result[link]?>><?= $result[titolo]?></
       a></td>
     <td><?= $result[autore]?></td>
    <td><?= $result[relatore]?></td>
    <td><?= $result[gruppo]?></td>
    <td><?= $result[data]?></td>
```

```
    </tr>
    <?php } ?>
   </tbody>
  </table>
 <?php }


/*
 * -------------------------------------------------------
 *  Funzione per la creazione della tabella Articoli
 *  -------------------------------------------------------
 */

function tabella_art(){ ?>
 <form class="bottone_upload" action="inserimento.php?
  upload=Articoli" method="post">
  <input type="submit" value="upload">
 </form>
 <form class="cerca" action="cerca.php?cerca=Articoli"
  method="post">
  <input type="submit" value="cerca">
 </form>
 <div align="center">
  <table>
   <thead><tr> <th>Articolo</th> <th>Autore</th> <th>Gruppo
      </th>
    <th>Data</th> </tr></thead>
   <?php

   /* Connette al database ODBC invocando i driver */

   $dsn = 'sqlite:C:\Program Files\Apache Software
    Foundation\Apache2.2\htdocs\database.sqlite';
   try {
    $dbh = new PDO($dsn);
   } catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
```

```
   }
   $dbh->beginTransaction();
   $query_a="SELECT * FROM Articoli";
   $prep_query=$dbh->query($query_a);
   $arr = $prep_query->fetchAll();
   $i = 0;
   foreach ($arr as $result){
        $i++;
        if ($i%2) {$riga = "dispari";}
        else {$riga = "pari";}
    ?>
    <tbody>
     <tr class="<?=$riga?>">
      <td><a href=<?=$result[link]?>><?= $result[titolo
         ]?></a></td>
      <td><?= $result[autore]?></td>
        <td><?= $result[gruppo]?></td>
        <td><?= $result[data]?></td>
     </tr>
    <?php } ?>
   </tbody>
  </table>
 <?php }


/*
 * -------------------------------------------------------
 *  Funzione per la creazione della tabella Documenti
 * -------------------------------------------------------
 */

function tabella_doc(){ ?>
 <form class="bottone_upload" action="inserimento.php?
  upload=Documenti" method="post">
  <input type="submit" value="upload">
 </form>
 <form class="cerca" action="cerca.php?cerca=Documenti"
```

```
 method="post">
 <input type="submit" value="cerca">
</form>
<div align="center">
 <table>
  <thead><tr> <th>Documento</th> <th>Autore</th> <th>
     Gruppo</th>
   <th>Data</th> </tr></thead>
  <?php

       /* Connette al database ODBC invocando i driver */
       $dsn = 'sqlite:C:\Program Files\Apache Software
      Foundation\Apache2.2\htdocs\database.sqlite';
       try {
        $dbh = new PDO($dsn);
       } catch (PDOException $e) {
          echo 'Connection failed: ' . $e->getMessage();
       }
       $dbh->beginTransaction();
       $query_a="SELECT * FROM Documenti";
       $prep_query=$dbh->query($query_a);
       $arr = $prep_query->fetchAll();
       $i = 0;
       foreach ($arr as $result){
               $i++;
               if ($i%2) {$riga = "dispari";}
               else {$riga = "pari";}
       ?>
       <tbody>
       <tr class="<?=$riga?>">
       <td><a href=<?=$result[link]?>><?= $result[titolo
          ]?></a></td>
       <td><?= $result[autore]?></td>
         <td><?= $result[gruppo]?></td>
         <td><?= $result[data]?></td>
       </tr>
    <?php } ?>
```

```
    </tbody>
   </table>
 <?php }

/*
 * -------------------------------------------------------
 * Funzione di ricerca all'interno delle tabelle
 * -------------------------------------------------------
 */

function cerca($type){

  /* Connette al database ODBC invocando i driver */
  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
        $dbh = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbh->beginTransaction();
      ?>
        <table>
         <?php
          echo "<BR>";
          $valore=$_POST['com'];
          $valore=ucfirst(strtolower($valore));
          $tab = $type;
          if ($tab=="Tesi") {
           $query_b="SELECT * FROM $tab WHERE (autore='
              $valore')
         or (titolo='$valore') or (relatore='$valore')
         or (gruppo='$valore')";
          }
          else {
           $query_b="SELECT * FROM $tab WHERE (autore='
              $valore')
```

```
         or (titolo='$valore') or (gruppo='$valore')";
          }
          $prep_query=$dbh->query($query_b);
          $arr = $prep_query->fetchAll();
          foreach ($arr as $result){
                $i++;
                if ($i%2) {$riga = "dispari";}
                else {$riga = "pari";}
          ?>
         <tbody>
          <tr class="<?=$riga?>">
         <td><a href=<?=$result[link]?>><?= $result[titolo
            ]?></a>
         </td>
           <td><?= $result[autore]?></td>
           <td><?= $result[relatore]?></td>
           <td><?= $result[gruppo]?></td>
          </tr>
       <?php } ?>
      </tbody>
     </table>
    <?php }

/*
 * ------------------------------------------------
 * Ritorna l'id dell'utente loggato oppure FALSE se
 * non lo e'
 * ------------------------------------------------
 */

function id_utente_loggato(){

 // tolgo le eventuali info sul vecchio login
 if (!isset($_SESSION['token']))
  return FALSE;
  $query_a=("SELECT * FROM authlogin WHERE
   (authcode='{$_SESSION['token']}')");
```

```
  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
   $dbg = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbg->beginTransaction();
  $prep_query=$dbg->query($query_a);
  $a = $prep_query->fetchAll();
  if ($a[0]['authcode'] == $_SESSION['token'] ){

   // siamo loggati

   return $a[0]['idutente'];
   } else
   return FALSE;
 }


/*
 * ------------------------------------------------
 *  Funzione per effettuare il logout
 * ------------------------------------------------
 */

function logout(){
 unset($_SESSION['token']);
 ?>
 <div class="infologin"> Logout eseguito </div>
 <meta http-equiv="refresh" content="0; url=SensiNet.php">
 <?php } ?>

  </div>
 </body>
</html>
```

8.3.6 cerca.php

```
<HTML>
 <HEAD>
  <link rel="stylesheet" href="stilehome.css" type="text/
     css" />
  <img class="logo" alt="logo_SensiNet" src="logoSensiNet.
     gif">
  <TITLE>SensNet</TITLE>
 </HEAD>
  <?php session_start();
   $idutente = id_utente_loggato();
   if (($idutente!=1)&&($idutente!=FALSE)){
        $type= $_GET['cerca'];

    if ($type=='Tesi') {
         cerca($type);
    }
    if ($type=='Articoli') {
         cerca($type);
    }
    if ($type=='Documenti') {
         cerca($type);
    }
   }?>

   <form class ="cerca" method="post">
        <input type="text" name="com">
          
        <input type="submit" value="cerca">
        <input type="hidden" name="tipo" value="<?= $type
           ?>">
   </form>
   <?php


/*
 * -------------------------------------------------
```

```
* Funzione di ricerca all'interno delle tabelle
* ------------------------------------------------
*/
  function cerca($type){

   /* Connette al database ODBC invocando i driver */

   $dsn = 'sqlite:C:\Program Files\Apache Software
    Foundation\Apache2.2\htdocs\database.sqlite';
   try {
    $dbh = new PDO($dsn);
   } catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
    }
   $dbh->beginTransaction();
   ?>
   <table>
       <thead><tr> <th>Tesi</th> <th>Autore</th> <th>
          Relatore</th>
      <th>Gruppo</th> <th>Data</th></tr></thead>

     <?php
        echo "<BR>";
        $valore=$_POST['com'];
        $valore=ucfirst(strtolower($valore));
        $tab = $type;
        if ($tab=="Tesi") {
          $query_b="SELECT * FROM $tab WHERE (autore='
             $valore')
        or (titolo='$valore') or (relatore='$valore') or
        (gruppo='$valore')";
        }
        else {
          $query_b="SELECT * FROM $tab WHERE (autore='
             $valore')
        or (titolo='$valore') or (gruppo='$valore')";
        }
```

```
          $prep_query=$dbh->query($query_b);
          $arr = $prep_query->fetchAll();
          foreach ($arr as $result){
          $i++;
          if ($i%2) {$riga = "dispari";}
          else {$riga = "pari";}
          ?>
          <tbody>
           <tr class="<?=$riga?>">
          <td><a href=<?=$result[link]?>><?= $result[titolo
             ]?></a>
          </td>
            <td><?= $result[autore]?></td>
            <td><?= $result[relatore]?></td>
            <td><?= $result[gruppo]?></td>
            <td><?= $result[data]?></td>
           </tr>
         <?php } ?>
       </tbody>
      </table>
     <?php }

/*
 * -------------------------------------------------
 * Ritorna l'id dell'utente loggato oppure FALSE se
 * non lo e'
 * -------------------------------------------------
 */

function id_utente_loggato(){

 // tolgo le eventuali info sul vecchio login
 if (!isset($_SESSION['token']))
  return FALSE;
  $query_a=("SELECT * FROM authlogin WHERE
   (authcode='{$_SESSION['token']}')");
```

```
  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
   $dbg = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbg->beginTransaction();
  $prep_query=$dbg->query($query_a);
  $a = $prep_query->fetchAll();
  if ($a[0]['authcode'] == $_SESSION['token'] ){

   // siamo loggati

   return $a[0]['idutente'];
   } else
   return FALSE;
 }


/*
 * ------------------------------------------------
 *  Funzione per effettuare il logout
 * ------------------------------------------------
 */

function logout(){
 unset($_SESSION['token']);
 ?>
 <div class="infologin"> Logout eseguito </div>
 <meta http-equiv="refresh" content="0; url=SensiNet.php">
 <?php } ?>

        </div>
      </body>
     </html>
     <body>
```

```
      <ul class="menu">
         <li><a href="home.php?tipo=Tesi">Tesi</a></li>
         <li><a href="home.php?tipo=Articoli">Articoli</a
            ></li>
         <li><a href="home.php?tipo=Documenti">
            Documentazioni</a></li>
         <li><a href="sensori.php">Sensori</a></li>
         <li><a href="home.php?tipo=logout">Logout</a></li>
        </ul>
  </body>
</HTML>
```

8.3.7 home.php

```
<html>
 <body>
 <link rel="stylesheet" href="stilehome.css" type="text/css
    " />
 <img class="logo" alt="logo_SensiNet" src="logoSensiNet.
    gif">
 <div class="time"><?php echo date("d-m-Y");?></div>
 <?PHP
  session_start();
  $idutente = id_utente_loggato();
  if ($idutente == FALSE) {
   ?>
   <div class="infologin"> Non hai i permessi
    per accedere al sito. Esegui il Login </div>

   <meta http-equiv="refresh" content="2; url=SensiNet.php
      ">
   <?php
  }
  if ($idutente == 1) {?>
   <meta http-equiv="refresh" content="2; url=Admin.php">
   <?php
  }
```

```
  else {
   ?>
   <ul class="menu">
    <li><a href="home.php?tipo=Tesi">Tesi</a></li>
    <li><a href="home.php?tipo=Articoli">Articoli</a></li>
    <li><a href="home.php?tipo=Documenti">Documentazioni</a
       ></li>
    <li><a href="sensori.php">Sensori</a></li>
    <li><a href="home.php?tipo=logout">Logout</a></li>
  </ul>

<?php

  /* Connette al database ODBC invocando i driver */

  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\dbsensori.sqlite';
  try {
   $dbh = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbh->beginTransaction();

  // Recupero i parametri da passare alla funzione
  // predefinita per l'upload

  $type= $_GET['tipo'];?>
  <div class="sezione"><?php echo 'Gestione '.$type;?>
  </div>
  <?php

   if ($type=='Tesi') {
    tabella_tesi();}

   if ($type=='Articoli') {
    tabella_art();}
```

```
   if ($type=='Documenti') {
    tabella_doc();}

   if ($type=='logout') logout();
  }



/*
 * ------------------------------------------
 *  Funzione per la creazione della tabella Tesi
 *  ------------------------------------------
 */

function tabella_tesi(){ ?>
 <form class="bottone_upload" action="inserimento.php?
    upload=Tesi"
   method="post">
   <input type="submit" value="upload">
 </form>
 <form class="cerca" action="cerca.php?cerca=Tesi" method="
    post">
   <input type="submit" value="cerca">
 </form>
 <table>
  <thead><tr> <th>Tesi</th> <th>Autore</th> <th>Relatore</
     th>
    <th>Gruppo</th> <th>Data</th></tr></thead>

  <?php

   /* Connette al database ODBC invocando i driver */
   $dsn = 'sqlite:C:\Program Files\Apache Software
    Foundation\Apache2.2\htdocs\database.sqlite';
   try {
    $dbh = new PDO($dsn);
```

```
   } catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
   }
   $dbh->beginTransaction();
   $query_a="SELECT * FROM Tesi";
   $prep_query=$dbh->query($query_a);
   $arr = $prep_query->fetchAll();
   $i = 0;
   foreach ($arr as $result){
        $i++;
        if ($i%2) {$riga = "dispari";}
        else {$riga = "pari";}
   ?>
   <tbody>
    <tr class="<?=$riga?>">
     <td><a href=<?=$result[link]?>><?= $result[titolo]?></
        a></td>
      <td><?= $result[autore]?></td>
     <td><?= $result[relatore]?></td>
     <td><?= $result[gruppo]?></td>
     <td><?= $result[data]?></td>
    </tr>
    <?php } ?>
   </tbody>
  </table>
 <?php }


/*
 * ------------------------------------------------------
 *  Funzione per la creazione della tabella Articoli
 *  ------------------------------------------------------
 */

function tabella_art(){ ?>
 <form class="bottone_upload" action="inserimento.php?
  upload=Articoli" method="post">
```

```
 <input type="submit" value="upload">
</form>
<form class="cerca" action="cerca.php?cerca=Articoli"
 method="post">
 <input type="submit" value="cerca">
</form>
<div align="center">
 <table>
  <thead><tr> <th>Articolo</th> <th>Autore</th> <th>Gruppo
     </th>
   <th>Data</th> </tr></thead>
  <?php

  /* Connette al database ODBC invocando i driver */

  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
   $dbh = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbh->beginTransaction();
  $query_a="SELECT * FROM Articoli";
  $prep_query=$dbh->query($query_a);
  $arr = $prep_query->fetchAll();
  $i = 0;
  foreach ($arr as $result){
       $i++;
       if ($i%2) {$riga = "dispari";}
       else {$riga = "pari";}
   ?>
   <tbody>
    <tr class="<?=$riga?>">
     <td><a href=<?=$result[link]?>><?= $result[titolo
        ]?></a></td>
     <td><?= $result[autore]?></td>
```

```
        <td><?= $result[gruppo]?></td>
        <td><?= $result[data]?></td>
     </tr>
    <?php } ?>
   </tbody>
  </table>
 <?php }


/*
 * -------------------------------------------------------
 *  Funzione per la creazione della tabella Documenti
 * -------------------------------------------------------
 */

function tabella_doc(){ ?>
 <form class="bottone_upload" action="inserimento.php?
  upload=Documenti" method="post">
  <input type="submit" value="upload">
 </form>
 <form class="cerca" action="cerca.php?cerca=Documenti"
  method="post">
  <input type="submit" value="cerca">
 </form>
 <div align="center">
  <table>
   <thead><tr> <th>Documento</th> <th>Autore</th> <th>
      Gruppo</th>
    <th>Data</th> </tr></thead>
   <?php

        /* Connette al database ODBC invocando i driver */
        $dsn = 'sqlite:C:\Program Files\Apache Software
       Foundation\Apache2.2\htdocs\database.sqlite';
        try {
         $dbh = new PDO($dsn);
        } catch (PDOException $e) {
```

```
            echo 'Connection failed: ' . $e->getMessage();
        }
        $dbh->beginTransaction();
        $query_a="SELECT * FROM Documenti";
        $prep_query=$dbh->query($query_a);
        $arr = $prep_query->fetchAll();
        $i = 0;
        foreach ($arr as $result){
                $i++;
                if ($i%2) {$riga = "dispari";}
                else {$riga = "pari";}
        ?>
        <tbody>
        <tr class="<?=$riga?>">
        <td><a href=<?=$result[link]?>><?= $result[titolo
           ]?></a></td>
        <td><?= $result[autore]?></td>
          <td><?= $result[gruppo]?></td>
          <td><?= $result[data]?></td>
        </tr>
     <?php } ?>
    </tbody>
   </table>
 <?php }

/*
 * -------------------------------------------------------
 * Funzione di ricerca all'interno delle tabelle
 * -------------------------------------------------------
 */

function cerca($type){

  /* Connette al database ODBC invocando i driver */
  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
```

```
      $dbh = new PDO($dsn);
} catch (PDOException $e) {
 echo 'Connection failed: ' . $e->getMessage();
}
$dbh->beginTransaction();
    ?>
      <table>
       <?php
        echo "<BR>";
        $valore=$_POST['com'];
        $valore=ucfirst(strtolower($valore));
        $tab = $type;
        if ($tab=="Tesi") {
         $query_b="SELECT * FROM $tab WHERE (autore='
            $valore')
       or (titolo='$valore') or (relatore='$valore')
       or (gruppo='$valore')";
        }
        else {
         $query_b="SELECT * FROM $tab WHERE (autore='
            $valore')
       or (titolo='$valore') or (gruppo='$valore')";
        }
        $prep_query=$dbh->query($query_b);
        $arr = $prep_query->fetchAll();
        foreach ($arr as $result){
              $i++;
              if ($i%2) {$riga = "dispari";}
              else {$riga = "pari";}
        ?>
       <tbody>
        <tr class="<?=$riga?>">
       <td><a href=<?=$result[link]?>><?= $result[titolo
          ]?></a>
       </td>
         <td><?= $result[autore]?></td>
         <td><?= $result[relatore]?></td>
```

```
            <td><?= $result[gruppo]?></td>
           </tr>
       <?php } ?>
      </tbody>
     </table>
    <?php }

/*
 * ------------------------------------------------
 * Ritorna l'id dell'utente loggato oppure FALSE se
 * non lo e'
 * ------------------------------------------------
 */

function id_utente_loggato(){

 // tolgo le eventuali info sul vecchio login
 if (!isset($_SESSION['token']))
  return FALSE;
  $query_a=("SELECT * FROM authlogin WHERE
   (authcode='{$_SESSION['token']}')");

  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
   $dbg = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbg->beginTransaction();
  $prep_query=$dbg->query($query_a);
  $a = $prep_query->fetchAll();
  if ($a[0]['authcode'] == $_SESSION['token'] ){

   // siamo loggati

   return $a[0]['idutente'];
```

```
   } else
   return FALSE;
 }


/*
 * ------------------------------------------------
 *  Funzione per effettuare il logout
 * ------------------------------------------------
 */

function logout(){
 unset($_SESSION['token']);
 ?>
 <div class="infologin"> Logout eseguito </div>
 <meta http-equiv="refresh" content="0; url=SensiNet.php">
 <?php } ?>

  </div>
 </body>
</html>
```

8.3.8 inserimento.php

```
<HTML>
 <HEAD>
 <link rel="stylesheet" href="stilehome.css" type="text/css
    " />
 <img class="logo" alt="logo_SensiNet" src="logoSensiNet.
    gif">
 <TITLE>SensiNet</TITLE>
 </HEAD>
 <body> <?
  ini_set("memory_limit","48M"); // TODO La soluzione
     temporanea
  session_start();
  $idutente = id_utente_loggato();
```

```
if ($idutente == FALSE) {
 ?>
 <div class="infologin"> Non hai i permessi per accedere
  al sito. Esegui il Login </div>

 <meta http-equiv="refresh" content="2; url=SensiNet.php
    ">
 <?php
 }
 if (($idutente!=1)&&($idutente!=FALSE)){
      $type= $_GET['upload'];

       if ($type=='Tesi') {
        $cartella = 'C:\Program Files\Apache Software
       Foundation\Apache2.2\htdocs\Tesi\\';
        upload_tesi($cartella);
       }
       if ($type=='Articoli') {
        $cartella = 'C:\Program Files\Apache Software
       Foundation\Apache2.2\htdocs\Articoli\\';
        upload_art($cartella);
       }
       if ($type=='Documenti') {
        $cartella = 'C:\Program Files\Apache Software
       Foundation\Apache2.2\htdocs\Documenti\\';
        upload_doc($cartella);
       }
 } else {?>
  <?php

      // Recupero tutti i dati mandati tramite POST
    //dalle altre pagine

      $nome=ucfirst(strtolower($_REQUEST['nome']));
      $cognome=ucfirst(strtolower($_REQUEST['cognome']));
      $username=$_REQUEST['username'];
      $passwd=md5($_REQUEST['passwd']);
```

```
 $ruolo=ucfirst(strtolower($_REQUEST['ruolo']));
 $relatore=ucfirst(strtolower($_REQUEST['relatore'])
    );
 $corelatore=ucfirst(strtolower($_REQUEST['
    corelatore']));
 $tel=ucfirst(strtolower($_REQUEST['tel']));
 $email=ucfirst(strtolower($_REQUEST['email']));
 $gruppo=ucfirst(strtolower($_REQUEST['gruppo']));
 $utente=$_REQUEST['utente_canc'];
 $tesi=$_REQUEST['tesi_canc'];
 $articolo=$_REQUEST['art_canc'];
 $doc=$_REQUEST['doc_canc'];

 $dsn = 'sqlite:C:\Program Files\Apache Software
Foundation\Apache2.2\htdocs\database.sqlite';
 try {
  $dbh = new PDO($dsn);
 } catch (PDOException $e) {
  echo 'Connection failed: ' . $e->getMessage();
 }
 $dbh->beginTransaction();

 /*
  * verifico se si vuole cancellare un utente,
  * una tesi ecc.. o se si vuole
  * aggiungere un nuovo utente
  */

 if (($utente!="") || ($utente=="Inserisci utente"))
     {
$query_a=("DELETE FROM utente WHERE (username= '{
   $utente}')");
  $prep_query=$dbh->exec($query_a);
  $dbh->commit();
 ?>
 <div class="infoLogin"> Cancellazione avvenuta con
    successo!
```

```
</div>

  <meta http-equiv="refresh" content="2; url=Admin.
     php?
 tipo=Utenti">
  <?php
  }
elseif (($tesi!="") || ($tesi=="Inserisci titolo")) {
   $query_a=("DELETE FROM Tesi WHERE (titolo= '{$tesi
      }')");
   $prep_query=$dbh->exec($query_a);
   $dbh->commit();
  ?>
  <div class="infoLogin"> Cancellazione avvenuta con
     successo!
</div>

  <meta http-equiv="refresh" content="2; url=Admin.
     php?
 tipo=Tesi">
  <?php
  }
elseif (($articolo!="") || ($articolo=="Inserisci
   titolo")) {
   $query_a=("DELETE FROM Articoli WHERE
  (titolo= '{$articolo}')");
   $prep_query=$dbh->exec($query_a);
   $dbh->commit();
   ?>
  <div class="infoLogin"> Cancellazione avvenuta con
     successo!
</div>
  <meta http-equiv="refresh" content="2; url=Admin.
     php?
 tipo=Articoli">
  <?php
  }
```

```
        elseif (($doc!="") || ($doc=="Inserisci titolo")) {
         $query_a=("DELETE FROM Documenti WHERE (titolo= '{
            $doc}')");
         $prep_query=$dbh->exec($query_a);
         $dbh->commit();
        ?>
        <div class="infoLogin"> Cancellazione avvenuta con
           successo!
      </div>
        <meta http-equiv="refresh" content="2; url=Admin.
           php?
       tipo=Documenti">
        <?php
        } else {
         $query_b=("INSERT INTO utente
        (nome,cognome,username,password,ruolo,relatore,
           corelatore,
        telefono,email,gruppo) VALUES ('{$nome}','{$cognome
           }',
        '{$username}','{$passwd}','{$ruolo}','{$relatore}',
        '{$corelatore}','{$tel}','{$email}','{$gruppo}')");

          $prep_query=$dbh->exec($query_b);
          $dbh->commit();
        ?>
        <div class="infoLogin"> Inserimento avvenuto con
           successo!
      </div>
        <meta http-equiv="refresh" content="2; url=Admin.
           php">
        <?php
        }
     }


/*
 * -------------------------------------------------
```

```
 * Ritorna l'id dell'utente loggato oppure FALSE se
 * non lo e'
 * ------------------------------------------------
 */

function id_utente_loggato(){

 // tolgo le eventuali info sul vecchio login
 if (!isset($_SESSION['token']))
  return FALSE;
  $query_a=("SELECT * FROM authlogin WHERE
   (authcode='{$_SESSION['token']}')");

  $dsn = 'sqlite:C:\Program Files\Apache Software
   Foundation\Apache2.2\htdocs\database.sqlite';
  try {
   $dbg = new PDO($dsn);
  } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
  }
  $dbg->beginTransaction();
  $prep_query=$dbg->query($query_a);
  $a = $prep_query->fetchAll();
  if ($a[0]['authcode'] == $_SESSION['token'] ){

   // siamo loggati

   return $a[0]['idutente'];
   } else
   return FALSE;
 }


/*
 * ------------------------------------------------
 * Funzione per l'upload tesi
 * ------------------------------------------------
```

```
 */

function upload_tesi($cartella){?>
 <form class="upload" method="post" enctype="multipart/form
    -data">
  <input type="file" name="miofile">
  <input type="submit" value="Upload">
 </form>
 <?php
  $percorso = $_FILES['miofile']['tmp_name'];
  $nome = $_FILES['miofile']['name'];

  // Eseguo l'upload controllando l'esito

  if (move_uploaded_file($percorso, $cartella . $nome)) {
   $dsn = 'sqlite:C:\Program Files\Apache Software
    Foundation\Apache2.2\htdocs\database.sqlite';
   try {
        $dbh = new PDO($dsn);
   } catch (PDOException $e) {
   echo 'Connection failed: ' . $e->getMessage();
   }
   $dbh->beginTransaction();
   $idutente = id_utente_loggato();
   $query_a=("SELECT cognome,gruppo,relatore FROM utente
    WHERE (idutente='$idutente')");
   $prep_query=$dbh->query($query_a);
   $arr = $prep_query->fetchAll();
   foreach ($arr as $result){
        $upnome=$result[cognome];
        $upgruppo=$result[gruppo];
        $uprelatore=$result[relatore];
   }
   $updata=date("d-m-Y");
   $link='"/Tesi/'.$nome.'"';

   $query_b=("INSERT INTO Tesi
```

```
   (titolo,autore,relatore,gruppo,data,link) VALUES
   ('{$nome}','{$upnome}','{$uprelatore}','{$upgruppo}',
    '{$updata}','{$link}')");

   prep_query=$dbh->exec($query_b);;
   $dbh->commit();
   ?>
   <div class="infoupload"> Inserimento avvenuto con
      successo! </div>
   <meta http-equiv="refresh" content="1; url=Home.php?tipo
      =Tesi">
  <?php
    }
  }

/*
 * -------------------------------------------------------
 *  Funzione per l'upload articoli
 * -------------------------------------------------------
 */

function upload_art($cartella){?>
 <form class="upload" method="post" enctype="multipart/form
    -data">
   <input type="file" name="miofile">
   <input type="submit" value="Upload">
 </form>
 <?php
  $percorso = $_FILES['miofile']['tmp_name'];
  $nome = $_FILES['miofile']['name'];
  print $_FILES;

  // Eseguo l'upload controllando l'esito

  if (move_uploaded_file($percorso, $cartella . $nome)) {
    $dsn = 'sqlite:C:\Program Files\Apache Software
     Foundation\Apache2.2\htdocs\database.sqlite';
```

```
    try {
        $dbh = new PDO($dsn);
    } catch (PDOException $e) {
     echo 'Connection failed: ' . $e->getMessage();
    }
    $dbh->beginTransaction();
    $idutente = id_utente_loggato();
    $query_a=("SELECT cognome,gruppo FROM utente WHERE
     (idutente='$idutente')");
    $prep_query=$dbh->query($query_a);
    $arr = $prep_query->fetchAll();
    foreach ($arr as $result){
        $upnome=$result[cognome];
        $upgruppo=$result[gruppo];
    }
    $updata=date("d-m-Y");
    $link='"/Articoli/'.$nome.'"';
    $query_b=("INSERT INTO Articoli (titolo,autore,gruppo,
       data,link)
     VALUES ('{$nome}','{$upnome}','{$upgruppo}','{$updata
        }',
     '{$link}')");

    $prep_query=$dbh->exec($query_b);;
    $dbh->commit();
    ?>
    <div class="infoupload"> Inserimento avvenuto con
       successo!
    </div>
    <meta http-equiv="refresh" content="1; url=Home.php?
     tipo=Articoli">
    <?php
    }
  }

/*
 * -------------------------------------------------
```

```
 *  Funzione per l'upload documenti
 *  ------------------------------------------------
 */

function upload_doc($cartella){?>
 <form class="upload" method="post" enctype="multipart/form
    -data">
   <input type="file" name="miofile">
   <input type="submit" value="Upload">
 </form>
 <?php
  $percorso = $_FILES['miofile']['tmp_name'];
  $nome = $_FILES['miofile']['name'];

  // Eseguo l'upload controllando l'esito

  if (move_uploaded_file($percorso, $cartella . $nome)) {
    $dsn = 'sqlite:C:\Program Files\Apache Software
     Foundation\Apache2.2\htdocs\database.sqlite';
    try {
        $dbh = new PDO($dsn);
    } catch (PDOException $e) {
     echo 'Connection failed: ' . $e->getMessage();
    }
    $dbh->beginTransaction();
   $idutente = id_utente_loggato();
   $query_a=("SELECT cognome,gruppo FROM utente WHERE
    (idutente='$idutente')");

   $prep_query=$dbh->query($query_a);
   $arr = $prep_query->fetchAll();
   foreach ($arr as $result){
        $upnome=$result[cognome];
        $upgruppo=$result[gruppo];
   }
   $updata=date("d-m-Y");
   $link='"/Documenti/'.$nome.'"';
```

```
   $query_b=("INSERT INTO Documenti (titolo,autore,gruppo,
      data,link)
    VALUES ('{$nome}','{$upnome}','{$upgruppo}',
    '{$updata}','{$link}')");

   $prep_query=$dbh->exec($query_b);;
   $dbh->commit();
   ?>
   <div class="infoupload"> Inserimento avvenuto con
      successo! </div>
   <meta http-equiv="refresh" content="1; url=Home.php?
    tipo=Documenti">
   <?php
     }
    }
   ?>
  </body>
</HTML>
```

Bibliografia

[1] J.A. Stankovic. *Wireless Sensor Networks*. 2006.

[2] Philip Levis. *TinyOS Programming*. 2006.

[3] S. Madden, W. Hong, J. Hellerstein, and M. Franklin. Tinydb web page. http://telegraph.cs.berkeley.edu/tinydb.

[4] D. Richard Hipp. Sqlite web page. http://www.sqlite.org/.

[5] Paul DuBois. *MySQL*. 2004. SLAU227C.

[6] Michael Stonebraker. Postgresql web page. http://www.postgresql.org/.

[7] Texas Instruments. *eZ430-RF2500 Development Tool*. Giugno 2008. SLAU227C.

[8] Chipcon. *CC2500 Single Chip Low Cost Low Power RF Transceiver*. 2006. SWRS040A.

[9] Texas Instruments. *MSP430x2xx Family 2008 Mixed Signal Products Users*. 2008. SLAU144E.

[10] Larry Friedman. *SimpliciTI Applications Guide*. Texas Instruments, Agosto 2008.

[11] Php web page. http://www.php.net/.