Università degli Studi di Milano

Dipartimento di Informatica
Laboratorio di Architetture Digitali

Progettazione e realizzazione di una stazione di rilevazione meteorologica

Giuseppe Fotino
Michele Tellarini

A.A. 2011/2012

Indice

Indice 2
Introduzione 3
Componenti 4
    Tmote 4
    Sensori Analogici 4
    Sensori Digitali 4
    Board 5
Installazione 9
    Installazione manuale 10
    Modalità chiosco 11
Centralina 13
    Struttura Hardware 14
    Funzionamento 15
Componenti software 35
    Cross-compilazione e installazione del firmware 40
Programmazione del Server 41
    Applicazione Java (LogStation) 41
    Database 50
    Applicazione Web 53
    Strumenti di monitoring 54
Bibliografia 56

Introduzione

La stazione meteorologica DI Weather nasce come progetto d'esame di Giuseppe Fotino e Michele Tellarini per il corso di Architetture Digitali tenuto dal professor Federico Pedersini. Obiettivo del progetto è la realizzazione di una stazione meteorologica di rilevamento dei dati atmosferici con l'ausilio di un scheda a microcontrollore collegata ad un server.

Compito del microcontrollore è leggere periodicamente i sensori e produrre un record da inviare al server per l'archiviazione in un database e la pubblicazione attraverso un'interfaccia web. Le piattaforme messe a confronto per la realizzazione della stazione meteorologica sono state l'ADUC7024, Arduino Uno e Tmote Sky: la scelta è ricaduta su Tmote per questioni di consumi inferiori, memoria flash da 1MB integrata, presenza di alcuni sensori on-board e ambiente di sviluppo.

Date le basi teoriche di architetture di elettronica digitale apprese durante il corso, il progetto consiste nell'affrontare e risolvere le problematiche relative alla progettazione di un sistema embedded: tra queste l'interfacciamento e il debug del dispositivo, l'uso e la conoscenza degli ambienti di sviluppo e dei linguaggi di programmazione (NesC, Java, SQL, php, JavaScript, HTML5) e non ultima una visione globale di progetto durante le fasi analitiche dello sviluppo.

La conclusione del lavoro ha visto un sistema dotato di termometri, igrometro, fotodiodi, barometro, pluviometro, anemometro e banderuola segnavento: tutti questi sensori sono stati collegati e gestiti dal Tmote Sky espanso con una basetta esterna per la connessione di tutti i sensori non integrati.

Durante la programmazione del Tmote e del server di logging è stato definito un protocollo di comunicazione tra i due sistemi, basato sulle primitive di comunicazione seriale messe a disposizione da TinyOs, che permette lo scambio dei dati meteo e di sincronizzazione. Il protocollo prevede un meccanismo tale per cui, in caso il server non sia più recettivo, i dati letti dal Tmote vengono momentaneamente archiviati nella memoria flash interna.

L'ultimo componente del sistema che è stato sviluppato è l'interfaccia web: un sito dal quale è possibile visualizzare in tempo reale le ultime letture e interrogare il database per analizzare lo storico dei dati meteorologici.
Il sito è ad oggi accessibile dall'indirizzo http://meteo.di.unimi.it.

Componenti

Segue un elenco sintetico dei componenti utilizzati e dei rispettivi datasheet.

Tmote

Datasheet: Tmote Sky

Sensori Analogici

Igrometro

- Modello: Sensirion SHT11
- Posizione: sul Tmote
- Note: combina un igrometro ed un termometro in un unico sensore. Gli strumenti possono essere letti separatamente ed è possibile correggere i valori di umidità con la temperatura.

Fotodiodo (luce visibile)

- Modello: Hamamatsu s1087
- Posizione: sul Tmote

Fotodiodo (radiazione infrarossa)

- Modello: Hamamatsu s1087-01
- Posizione: sul Tmote

Termometro

- Modello: AD592
- Posizione: sulla basetta
- Interfaccia: ADC1

Barometro

- Modello: MPXA6115A
- Posizione: sulla basetta
- Interfaccia: ADC0

Banderuola

- Modello: Argent Data Systems 80422
- Posizione: sostegno esterno
- Interfaccia: ADC2

Sensori Digitali

Anemometro

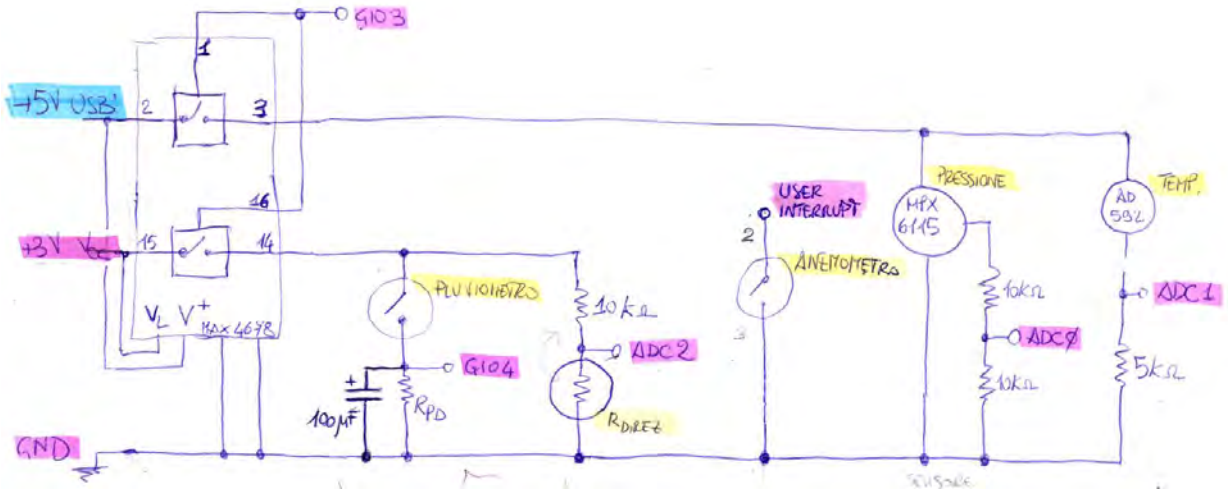
- Modello: Argent Data Systems 80422
- Posizione: sostegno esterno
- Interfaccia: User Interrupt

Pluviometro

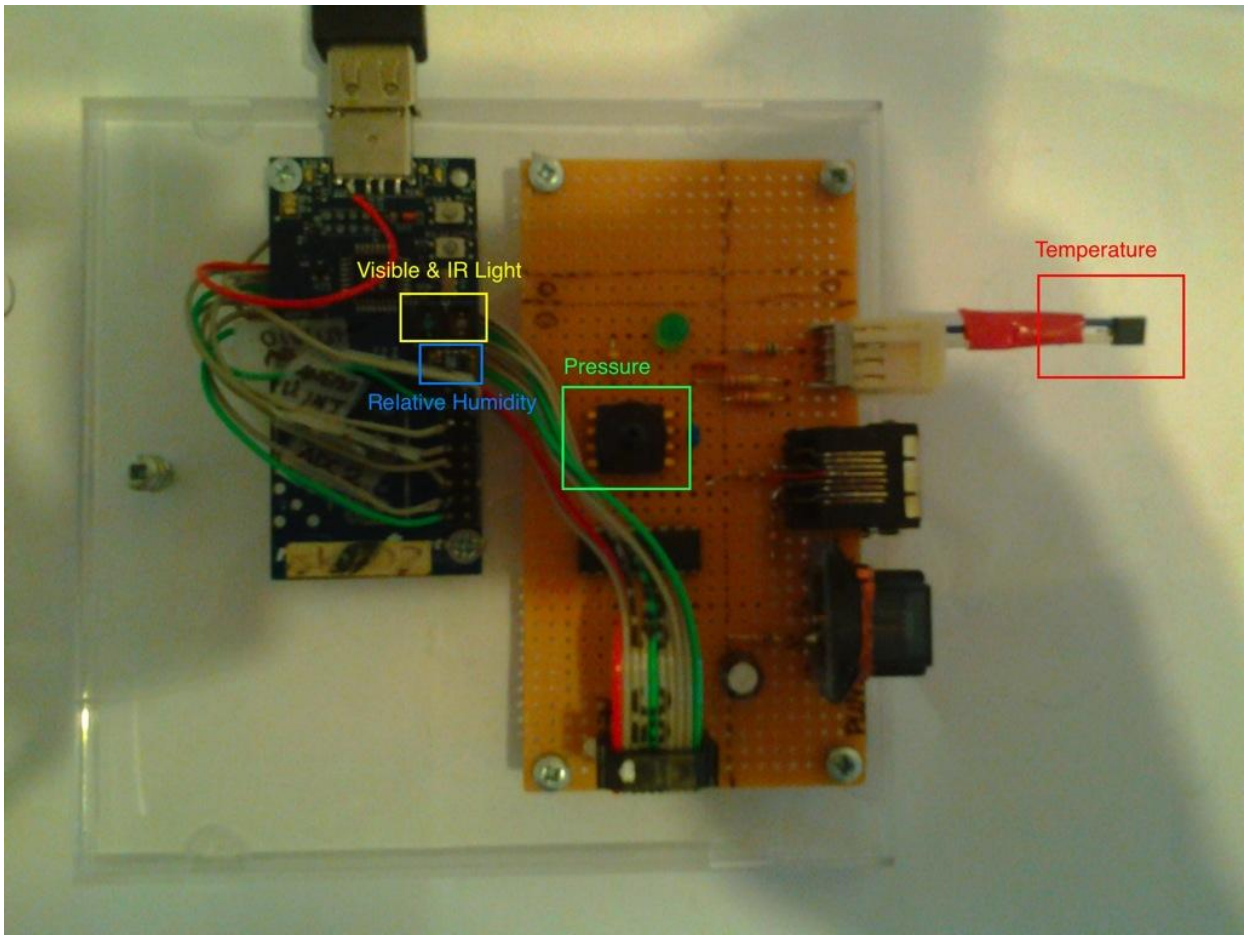
- Modello: [Argent Data Systems 80422]
- Posizione: sostegno esterno
- Interfaccia GIO2 (nello schema elettrico allegato è erroneamente collegato a GIO4, il codice lo collega alla `porta 23`, che secondo il datasheet del Tmote corrisponde proprio a GIO2)

Board

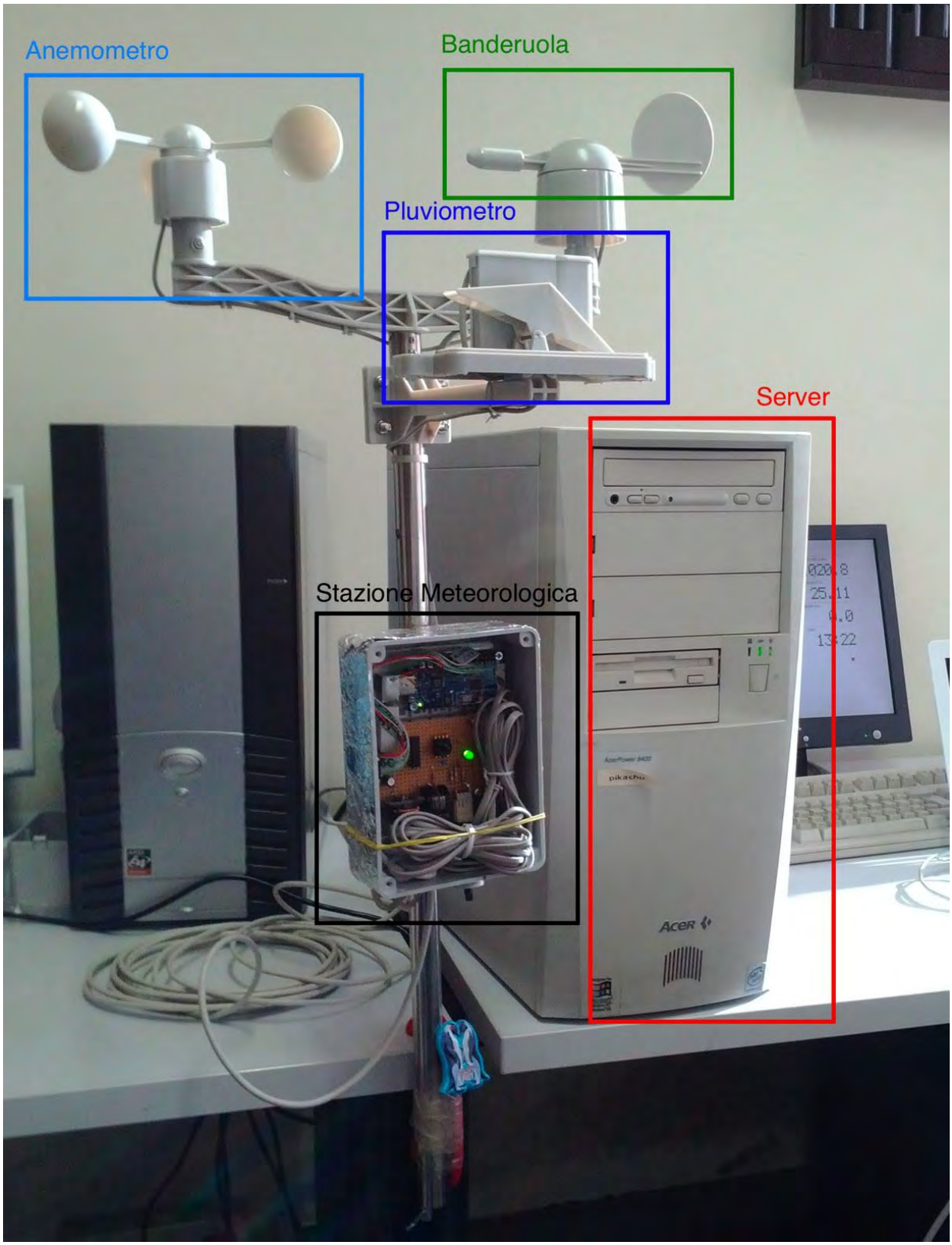
Schema elettrico della basetta.



Fotografia della basetta.



Il sistema assemblato in fase di testing.



La stazione meteorologica in funzione.

Installazione

Per l'installazione del sistema sono necessari i seguenti requisiti software:

- Java
- MySQL
- Java MySQL Connector (libmysql-java)
- Apache
- PHP

Per la "modalità chiosco" sono necessari anche:

- python
- python-webkit
- mingetty
- xorg

Strumenti di monitoring

- monit
- sendemail

Il sistema è stato installato su una macchina Ubuntu Linux versione 11.10 server; sebbene non testato, dovrebbe funzionare anche con altre distribuzioni Linux.

Per installare il sistema base (software di logging e sito web) è sufficiente decomprimere l'archivio `weather.tar.bz2` in una directory qualsiasi (anche la home va bene) ed eseguire con privilegi amministrativi lo script **`install.sh`**: questo script si occupa di configurare le varie componenti e di creare tutti i collegamenti necessari per avviare automaticamente la LogStation al boot del sistema.

Dopo questa fase di installazione nel sistema è presente un servizio chiamato `weather`. Per l'avvio e l'interruzione è conforme a tutti i servizi Linux:

```
sudo /etc/init.d/weather [start|stop]
```

Attenzione!
Una volta eseguita l'installazione non è possibile spostare i file, in particolare le classi Java della LogStation, in quanto i link punteranno tutti a quella directory.

Nota bene
Il sistema attualmente configurato prevede un servizio di monitoring che si occupa, tra le altre cose, di riavviare la LogStation in caso di failure. Per questo motivo lo stop del servizio `weather` deve essere eseguito successivamente allo stop del servizio `monit`[1]: in caso contrario il servizio `weather` sarà riavviato automaticamente dopo circa due minuti.

[1] Per maggiori dettagli su Monit leggere la sezione "Strumenti di monitoring"

Installazione manuale

Per eseguire un'installazione manuale, per prima cosa bisogna installare il pacchetto tinyos-2.1.1 reperibile sul repository http://hinrg.cs.jhu.edu.tinyos: per fare questo è necessario inserire la riga

```
deb http://tinyurl.com/cjyp6bm natty main
```

nel file `/etc/apt/sources.list` ed eseguire i comandi

```
sudo apt-get update
sudo apt-get install tinyos-2.1.1
```

Una volta completata l'installazione di questo pacchetto, eseguire:

```
bash /opt/tinyos-2.1.1/tinyos.sh
tos-install-jni
```

Verificata la presenza di MySQL sulla macchina è necessario creare il database e le due utenze: la prima serve per la LogStation e deve avere privilegi elevati, la seconda è dedicata al web server e necessita dei soli permessi di lettura.
Nella directory `source` (è tra i file che vengono creati scompattando l'archivio `weather.tar.bz2`) è presente il file `mysql.sql` che effettua queste operazioni automaticamente; per utilizzarlo eseguire:

```
mysql -u root < source/mysql.sql
```

Nota Bene
Per semplicità è indicato l'utente *`root`* di MySQL, ma i comandi del file `mysql.sql` possono essere eseguiti con successo da qualunque utente dotato dei privilegi per la creazione di un nuovo database e degli utenti con i relativi permessi.

Nota Bene
Il file `mysql.sql` creerà un nuovo database vuoto: la creazione della struttura della base di dati sarà effettuata automaticamente dalla LogStation al primo avvio.

A questo punto, se i componenti elencati nei requisiti sono stati tutti installati correttamente, è possibile eseguire la LogStation digitando

```
java LogStation
```

nella sottocartella `bin` del progetto. Se le operazioni precedenti hanno avuto buon esito è ora possibile vedere i dati dei sensori apparire a schermo ogni 5 secondi.

Attenzione!
Prima di eseguire la LogStation verificare il file **`bin/LogStation.config`**: l'indirizzo dell'istanza MySQL e le credenziali di accesso devono essere corrette.

Applicazione web

Per il funzionamento dell'applicazione web, è necessario installare sul server Apache2, php e il modulo mysql di php.
Una volta scompattato l'archivio `weather.tar.bz2` è sufficiente copiare tutti i file presenti in `www` nella directory di pubblicazione del web server (solitamente `/var/www/`).

Firewall

Poiché il server `meteo.ricerca.dsi.unimi.it` è esposto ad Internet, si è optato per la creazione di regole del firewall `iptables` della macchina per aumentarne la sicurezza.
Le regole sono presenti nel file `source/weather-iptables` dell'archivio `weather.tar.bz2`; non è detto che tali regole siano applicabili anche su altri server: è necessario configurare `iptables` a seconda delle impostazioni di rete e dei servizi presenti sulla macchina.

Attenzione!

Qualora si volesse riattivare tale setup è necessario caricare le regole ad ogni avvio del server: chiamare iptables una sola volta non memorizzerà le impostazioni al successivo riavvio del sistema. Uno dei metodi preferibili per questa operazione è copiare lo script `source/weather-iptables` in `/etc/network/if-up.d/`.

Modalità chiosco

La modalità chiosco trasforma il server in una stazione meteorologica completa di interfaccia video che riporta il pannello delle rilevazioni istantanee. Per configurare questa modalità è ovviamente necessario che il server sia collegato ad un monitor.

L'interfaccia configurata sulla macchina del dipartimento di informatica è tale per cui l'esecuzione di qualsiasi altro comando richiede la conoscenza di un utenza del server e relativa password. In caso di terminazione dell'interfaccia tramite **`CTRL + C`** la macchina interpreterà l'azione come un tentativo di shutdown e completerà l'operazione di spegnimento.

La configurazione del sistema è strettamente legata alla distribuzione Linux usata, in questo caso riporteremo le operazioni effettuate su Ubuntu linux 11.10 server.

Come prima cosa è stato installato `mingetty` e configurato su `tty1` modificando il file `/etc/init/tty1.conf` commentando la riga:

```
#exec /sbin/getty -8 38400 tty1
```

e aggiungendo

```
exec /sbin/mingetty --autologin <nomeutente> tty1
```

dove ovviamente `<nomeutente>` corrisponde ad un'utenza della macchina.
Dopo questa modifica, la macchina, terminata la fase di avvio, si loggerà automaticamente con l'utente specificato.

È quindi necessario modificare il file `.xinitrc` all'interno della home dell'utente specificato aggiungendo la riga:

```
source/pywebkitgtk-1.1.8/demos/mybrowser.py
```

Nello specifico deve essere trascritto il percorso completo al file `mybrowser.py` che può essere collocato in altre directory a patto che nella stessa cartella sia presente anche il file `inspector.py`.
Per verificare la correttezza del file `.xinitrc` è sufficiente avviare il server X con il comando

```
startx
```

Se con X parte anche il browser allora la modifica è corretta. Per modificare la dimensione della pagina web mostrata su X modificare la riga 343 del file mybrowser.py in base alla risoluzione impostata in `xorg.conf`.

Ultimo passo è l'avvio automatico di xorg: modificare il file `.bashrc` nella home dell'utente specificato inserendo in coda al file le righe di codice che seguono:

```
if [$(tty) == "/dev/tty1" ]; then
      startx;
      sudo shutdown -h now
fi;
```

La riga 3 (sudo...) è quella che determina lo spegnimento del server dopo la chiusura di X (quindi del pannello rilevazioni). È opzionale, ma se la si vuole inserire è necessario verificare che l'utente sia tra i `sudoers` per l'eseguibile `shutdown` altrimenti verrà chiesta la password di root per effettuare lo spegnimento, vanificando quindi l'effetto.

Nota Bene
Poiché l'esecuzione di Xorg è molto onerosa dal punto di vista dell'occupazione di memoria, per migliorare le prestazioni del server, al momento la modalità chiosco non viene avviata in maniera automatica al boot: il pannello a video parte al login dell'utente sul primo terminale (tty1).
In ogni caso, se dovesse essere necessario intervenire direttamente sul server, è possibile collegarsi da tty2. Per riabilitare l'avvio automatico della modalità chiosco è sufficiente modificare il file `/etc/init/tty1.conf` commentando la riga 10:

```
exec /sbin/getty -8 38400 tty1
```

e aggiungendo la seguente

```
exec /sbin/mingetty --autologin weather tty1
```

che attualmente è commentata.

Centralina

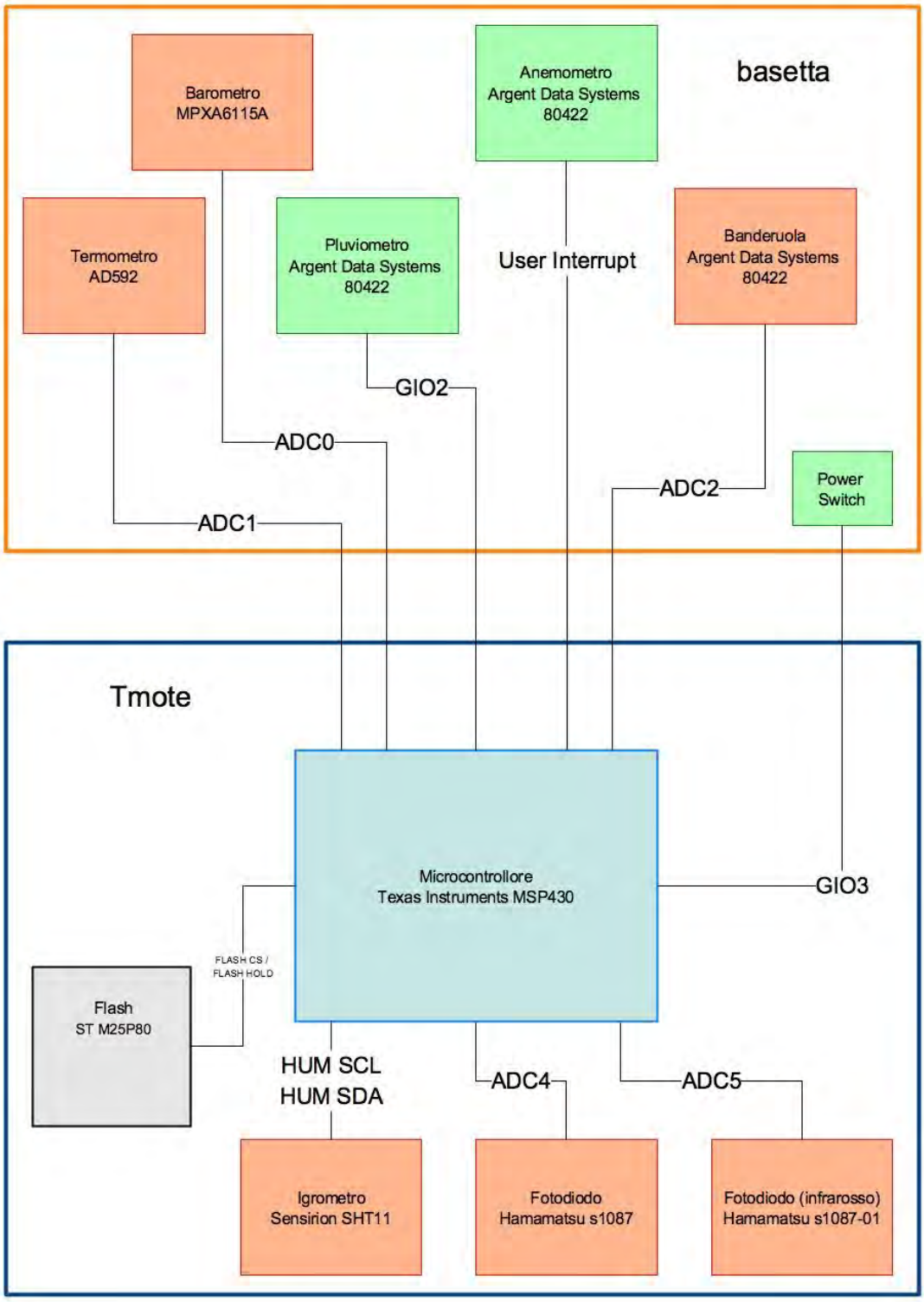
Il Tmote è stato programmato in ambiente Linux sfruttando Yeti2, un plugin per Eclipse che incorpora tutti gli strumenti necessari per la compilazione e l'installazione del codice sul dispositivo; il linguaggio utilizzato è il NesC: si tratta del linguaggio con il quale è stato sviluppato TinyOS, il sistema operativo che fornisce le librerie necessarie per l'accesso a tutte le risorse del dispositivo.

Riportiamo il documento di reference del linguaggio NesC e il manuale delle librerie disponibili in TinyOS.

http://nescc.sourceforge.net/papers/nesc-ref.pdf
http://www.tinyos.net/tinyos-2.1.0/doc/nesdoc/telosb/

Il secondo link punta alla reference per la piattaforma `telosb`, quella della versione del Tmote Sky in dotazione.

Struttura Hardware[2]

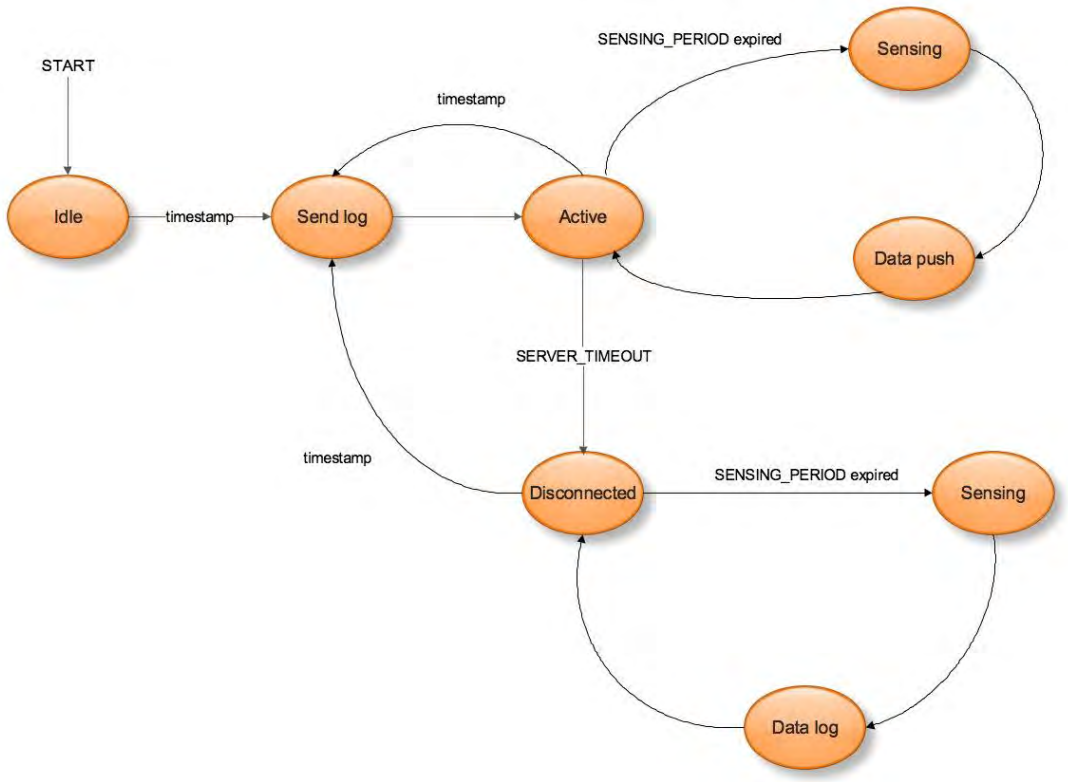


[2] con uno sfondo arancio sono evidenziati i sensori analogici; uno sfondo verde caratterizza quelli digitali

Funzionamento

Dopo l'accensione, il Tmote rimane in uno stato *idle* fino a quando non riceve un messaggio di sincronizzazione dal server: tale messaggio contiene il timestamp corrente ed è utilizzato, in combinazione con il timer interno, per determinare l'orario delle rilevazioni; segue una fase di invio del contenuto della memoria flash, su cui torneremo in seguito.
Lo stato *active* è lo stato di funzionamento normale del dispositivo: allo scadere di un timer, il Tmote avvia le operazioni di lettura dei sensori e invia (push) le rilevazioni al server; la ricezione di un nuovo messaggio di sincronizzazione durante lo stato *active* causerà un nuovo tentativo di lettura dei log: in condizioni normali la memoria flash non contiene alcun dato, quindi il Tmote ritornerà immediatamente nello stato *active*.
Il ciclo *active - send log* si ripete fino a quando la centralina continua a ricevere i messaggi di sincronizzazione dal server; se per qualsiasi motivo il riferimento temporale non arriva entro lo scadere di un certo tempo limite, il dispositivo passa nello stato *disconnected*.
In modalità *disconnected* il Tmote continuerà ad eseguire periodicamente le operazioni di lettura dei sensori, tuttavia le rilevazioni non saranno inviate subito al server (che potrebbe essere non raggiungibile), bensì saranno salvate in un log sulla memoria flash. Quando il server segnalerà nuovamente la sua presenza con l'invio di un timestamp aggiornato, il Tmote comunicherà tutti i dati presenti nel log prima di ritornare nello stato *active*.
La fase di invio dei log è presente anche dopo la ricezione del primo timestamp perché il dispositivo potrebbe aver salvato dei dati nella flash in una fase di disconnessione precedente al suo spegnimento.

Quello che segue è il diagramma degli stati della centralina meteo.



Alimentazione della centralina

Per il suo funzionamento, il Tmote richiede una tensione di alimentazione di 3V (vedi datasheet) che viene fornita dal dal cavo USB che collega il dispositivo al server.
Come si può vedere dallo schema elettrico, alcuni sensori montati sulla basetta esterna (termometro e barometro) devono essere alimentati con una tensione di 5V, anche questa presa dal cavo USB, mentre altri (pluviometro e banderuola) necessitano di una tensione inferiore, pari a soli 3V, che è invece fornita dal Tmote.
I sensori esterni sono collegati ad un interruttore (MAX4678) che a sua volta è pilotato da un segnale digitale proveniente dal Tmote: in questo modo il software può accendere la basetta esterna solo durante le operazioni di lettura dei sensori, risparmiando energia (una *feature* utile qualora si pensasse di dotare la centralina di una fonte di alimentazione diversa dal cavo, per esempio delle batterie e/o dei piccoli pannelli solari).
Fa eccezione l'anemometro che richiede solo un collegamento a GND poiché si comporta come un interruttore che ad ogni giro delle palette chiude il circuito collegato al piedino di interrupt.

Fase di avvio del sistema

All'avvio, TinyOS scatena l'evento `booted` dell'interfaccia `Boot`.
Nel modulo `wtCore` viene implementato l'*event handler* per `booted` che esegue le chiamate

```
call wtSensorsIf.init();
call wtServerInterface.StartServer();
```

la prima inizializza il modulo `wtSensors`, che si occupa della gestione e della lettura dei sensori; la seconda attiva il modulo che si occupa della comunicazione tra il Tmote ed il computer.

Segue il corpo del comando `init()` del modulo `wtSensors`.

```
command error_t wtSensorsIf.init() {
      call PowerSwitch.makeOutput();
      call PowerSwitch.set();
      call WindSpeed.init();
      call Rain.init();

      extTempBuffer  = malloc(BUFFER_SIZE * sizeof(uint16_t));
      pressureBuffer = malloc(BUFFER_SIZE * sizeof(uint16_t));
      return SUCCESS;
}
```

Le prime due chiamate servono per accendere la basetta che ospita i sensori "esterni", mentre le due chiamate successive inizializzano i componenti che gestiscono l'anemometro ed il pluviometro.

Il modulo `wtSensors` sfrutta l'interfaccia `GeneralIO`, istanziando un componente che viene chiamato `PowerSwitch`

```
interface GeneralIO as PowerSwitch;
```

Nel modulo principale, `AppConfig`, a `PowerSwitch` è assegnato il controllo della porta `GPIO3`, l'interfaccia digitale di I/O numero 3 del sistema: riprendendo lo schema elettrico della basetta si può notare che tale porta controlla l'interruttore per l'alimentazione della circuiteria relativa ai sensori.

```
PowerSwitch -> HplMsp430GeneralIOC.Port26;
```

Nel corpo di `init()` sono presenti due chiamate a due metodi diversi di `PowerSwitch`, `makeOutput()` e `set()`: il primo si occupa di configurare la porta in modo che il sistema possa scrivervi un bit (al contrario, il metodo `makeInput()` configura la porta in lettura), mentre il secondo scrive un "`1`" sull'uscita digitale. Queste operazioni hanno come effetto l'accensione della basetta esterna e l'alimentazione dei sensori ivi presenti; l'alimentazione del circuito è segnalata anche dall'accensione di un led verde posto sulla millefori.

Nota
L'interfaccia `GeneralIo` mette a disposizione anche il metodo `clr()` per inviare un segnale basso su un'uscita digitale.
Scrivere uno "`0`" sull'uscita che controlla l'alimentazione della basetta avrebbe come effetto lo spegnimento di tutta la circuiteria esterna; questo può tornare utile in un contesto in cui il sistema non è alimentato dalla rete elettrica, ma funziona a batterie: in tal caso è possibile programmare dei cicli di accensione e spegnimento dei sensori esterni, risparmiando energia.

Le chiamate successive nel corpo di `wtSensor.init()` inizializzano i componenti che gestiscono l'anemometro ed il pluviometro.

La procedura di inizializzazione dell'anemometro è molto semplice. Come possiamo vedere dallo schema elettrico della basetta esso è collegato alla porta `UserINT` in questo modo:

```
Wind -> HplMsp430InterruptC.Port27;
```

Poiché il funzionamento dell'anemometro prevede l'invio di un impulso ad ogni rotazione, l'inizializzazione del componente prevede solamente il `setup` della porta e l'azzeramento del contatore parziale di impulsi (e quindi giri delle palette).

```
command void wtWindSpeedIf.init(){
      turns = 0;
      call GpioInterrupt.enableRisingEdge();
}
```

La porta scatena un segnale di interrupt solamente sui fronti di salita dell'impulso, al fine di evitare falsi positivi o deadlock in caso di errori di precisione meccanica dell'anemometro in concomitanza con assenza di vento.

Il metodo `Rain.init()` che inizializza il pluviometro è un po' più complesso: il problema è dovuto alla struttura del sensore che, come l'anemometro genera dei segnali di interrupt. Data l'assenza di ulteriori porte utilizzabili come `UserINT`, esso è stato collegato al connettore `GPIO2` ed è stato implementato un meccanismo di *polling* in lettura; per prolungare per un breve

periodo la durata dell'impulso del pluviometro, alla basetta è stato collegato un condensatore da 100 µF.

```
command void wtRainIf.init(){
      rain = rainStatus = 0;
      call Rain.makeInput();
      call RainTimer.startPeriodic(POLLING_PERIOD);
}
```

Da notare la definizione di un timer per il polling del condensatore (`POLLING_PERIOD` è una costante che vale 200 ms) e di un contatore `rain` degli impulsi ricevuti dal sensore.

Le ultime due operazioni eseguite durante l'inizializzazione di `wtSensors` riguardano l'allocazione di memoria per le variabili

```
extTempBuffer  = malloc(BUFFER_SIZE * sizeof(uint16_t));
pressureBuffer = malloc(BUFFER_SIZE * sizeof(uint16_t));
```

Si tratta di due buffer che vengono utilizzati per filtrare, **già all'interno della centralina**, il rumore presente sulle letture provenienti dai sensori di temperatura e pressione ospitati sulla basetta esterna. `BUFFER_SIZE` è una costante definita nel file `settings.h` che vale 12: in questo modo il firmware compensa il rumore mediando 12 letture.

Abbiamo così concluso la trattazione della fase di avvio del componente `wtSensors`.

Consideriamo ora la seconda chiamata presente nel corpo di `wtCore.booted()`, ovvero `wtServer.StartServer()`, che avvia i servizi di comunicazione del sistema.

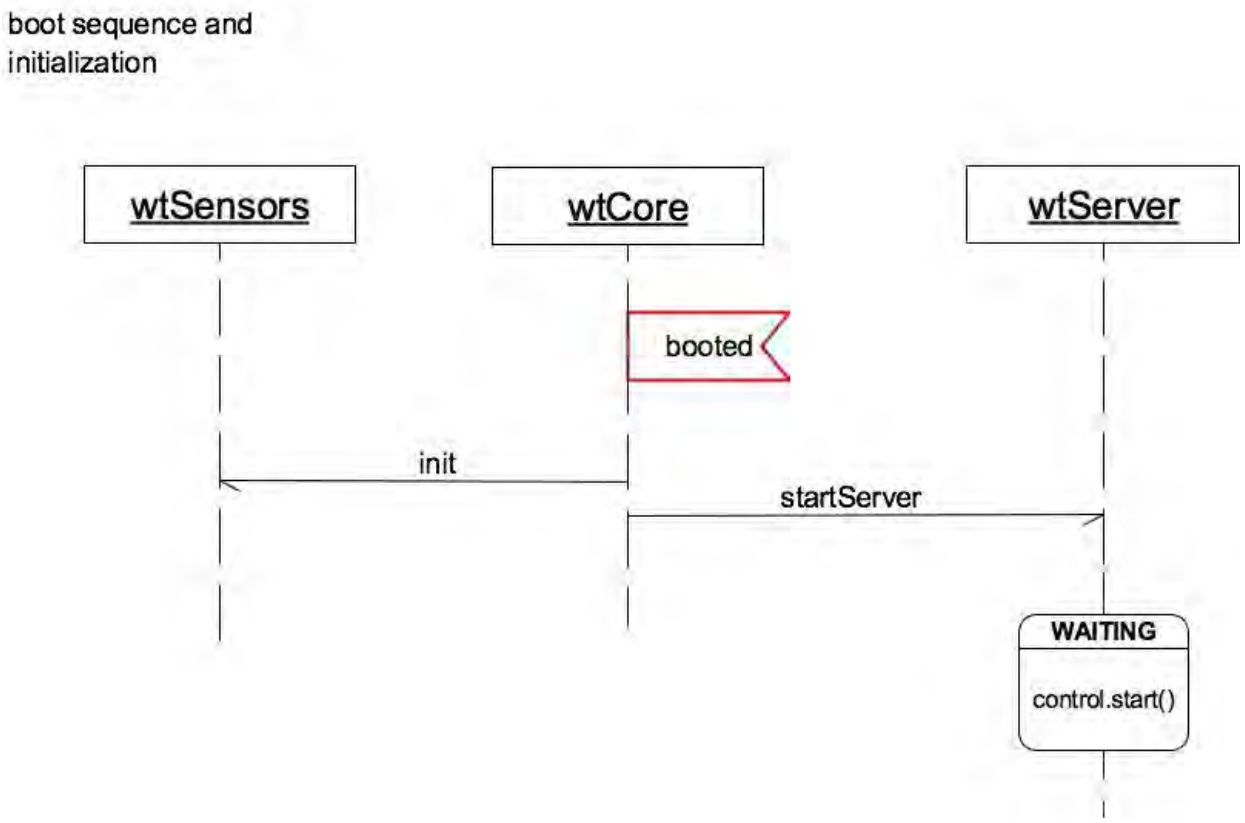
```
command void wtServerInterface.StartServer(){
      status = WAITING;
      call Control.start();
}
```

Il corpo del metodo introduce l'interfaccia `Control`: essa è connesso con il componente `SerialActiveMessageC.SplitControl` che si occupa della gestione del protocollo di comunicazione a basso livello tra la centralina e il server; la chiamata `start()` non fa altro che avviare tale componente di sistema.

A questo punto il sistema ha concluso la fase di inizializzazione post boot, quindi rimane in uno stato `idle` in attesa di una comunicazione da parte del server: ciò significa che i sensori sono pronti per le letture, ma queste non sono eseguite perché non è possibile assegnare un riferimento temporale ai dati acquisiti.
Come spiegato nel capitolo successivo, la fase di `idle` viene interrotta dalla ricezione di un messaggio da parte del server contenente un timestamp con il quale il Tmote allinea l'orologio interno da usare per memorizzare l'orario relativo a ciascuna lettura. La necessità di aggiungere un riferimento temporale alle rilevazioni è emersa quando si è presa in considerazione la possibilità che il server non sia attivo per un certo periodo di tempo o nella previsione di un

sistema che trasmettesse i dati via radio: in caso di mancata comunicazione con il computer, la centralina memorizza all'interno della propria memoria flash tutte le letture, in attesa di ristabilire il collegamento. In questo caso è ovvio che i dati devono essere arricchi con un timestamp.

Nell'immagine seguente è schematizzato il diagramma della fase di boot del sistema e la gestione del modulo `wtServer` nella fase di inizializzazione (attesa del primo timestamp del server e attivazione del timer che scatena la lettura ogni 5 secondi dei dati meteorologici).



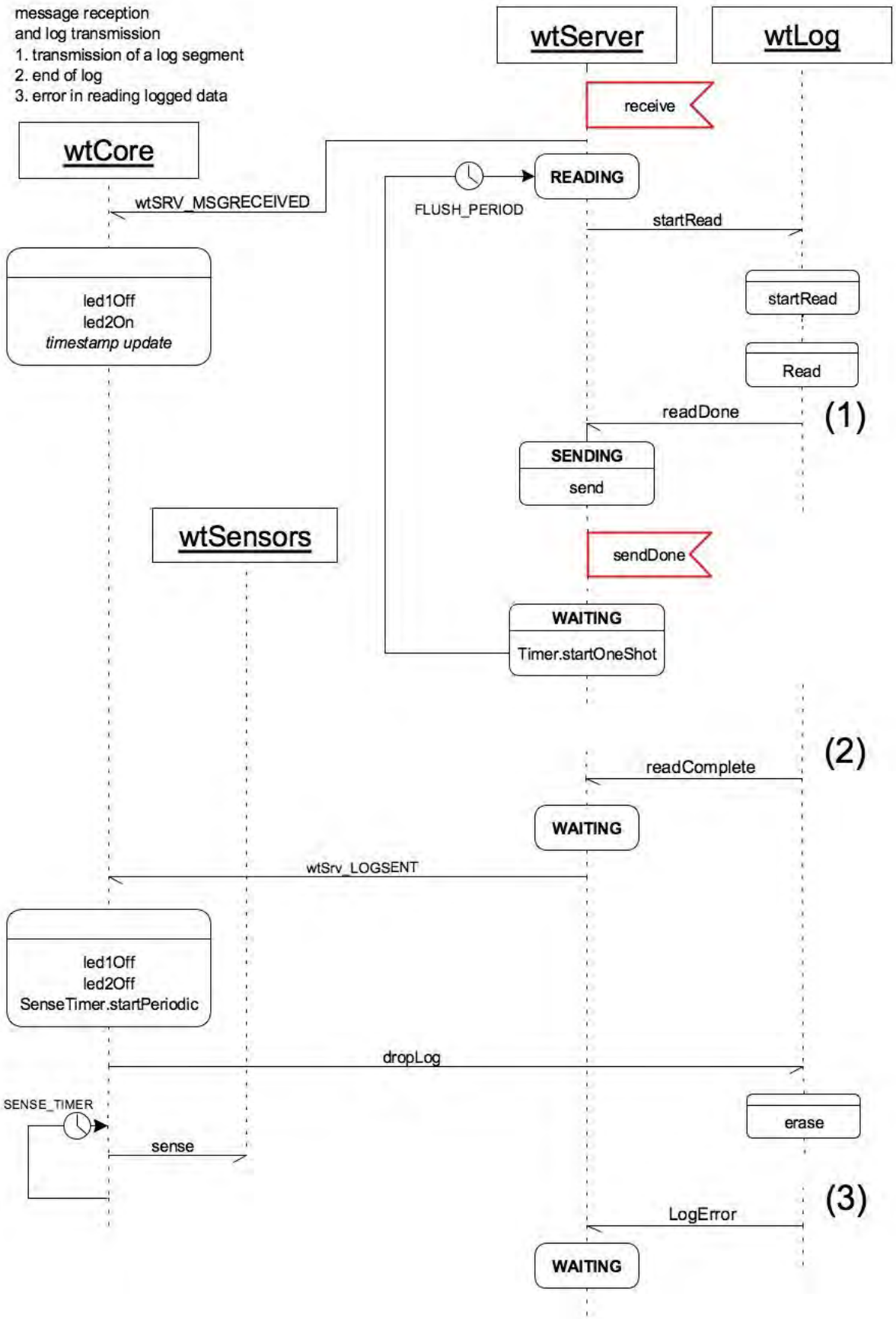
Alla ricezione di un messaggio l'interfaccia `Receive` scatena l'evento `receive`, che viene gestito in questo modo

```
event message_t * Receive.receive(message_t *msg, void *payload, uint8_t
len){

      if (len == sizeof(wt_syncRecord)) {
            s = (wt_syncRecord *) payload;
      }

      signal wtServerInterface.EventTriggered(wtSrv_MSGRECEIVED, s);

      if (call Log.StartRead() != SUCCESS){
            signal Log.LogError();
            signal wtServerInterface.EventTriggered(wtSrv_SENTABORTED, NULL);
      }
      return msg;
}
```

L'*event handler* effettua diverse operazioni; la prima è un cast del *payload* del pacchetto ricevuto alla struttura `wt_syncRecord` che rappresenta il messaggio di sincronizzazione; in seguito notifica al modulo `wtCore` la ricezione del messaggio, allegandolo.
Quando il modulo `wtCore` riceve la notifica del messaggio, sincronizza l'orologio interno

```
event void wtServerInterface.EventTriggered
      (uint16_t eventType, wt_syncRecord * record){

      if (eventType == wtSrv_MSGRECEIVED) {
            call Leds.led2On();
            call Leds.led1Off();

            lastMsgArrTime = call SenseTimer.getNow();

            if(record != NULL) {
                  lastTimeStamp = record->timestamp;
            }
      }

      [...]
}
```

Le prime due istruzioni all'interno del blocco if sono istruzioni di debug, che segnalano attraverso i led (spegnimento del led rosso e accensione del led blu) la ricezione del messaggio; seguono le due istruzioni fondamentali: la memorizzazione dell'orario interno di ricezione del timestamp e del timestamp stesso. Con queste due informazioni la centralina può correggere il proprio orologio e memorizzare/inviare pacchetti dati con un riferimento temporale corretto (relativamente al server).


message reception
and log transmission
1. transmission of a log segment
2. end of log
3. error in reading logged data
wtServer
wtLog
receive
wtCore
READING
wtSRV_MSGRECEIVED
FLUSH_PERIOD
startRead
led1Off
led2On
timestamp update
startRead
Read
readDone
(1)
SENDING
send
wtSensors
sendDone
WAITING
Timer.startOneShot
(2)
readComplete
WAITING
wtSrv_LOGSENT
led1Off
led2Off
SenseTimer.startPeriodic
dropLog
SENSE_TIMER
erase
sense
(3)
LogError
WAITING

Dopo avere notificato `wtCore` per il *time-sync*, `wtServer` avvia la lettura della memoria flash per scaricare eventuali buffer memorizzati e non ancora inviati, invocando il comando `Log.StartRead()`.

Il comando `Log.startRead()` non è altro che un wrapper per il comando `read()` dell'interfaccia `LogRead` del componente `LogStorageC` che gestisce la memoria flash.

```
command error_t wtLogInterface.StartRead() {
      return call LogRead.read(readRecord, sizeof(wt_logRecord));
}
```

Al termine delle operazioni di lettura, `LogRead` segnala un evento `readDone`; riportiamo il codice dell'*event handler* com'è dichiarato nel modulo `wtLog`

```
event void LogRead.readDone
      (void *buf, storage_len_t len, error_t error){

      if (len == sizeof(wt_logRecord) && error == SUCCESS){
            if(readRecord != NULL)
                  free(readRecord);

            readRecord = malloc(sizeof(wt_logRecord));
            memcpy(readRecord, buf, len);
            signal wtLogInterface.ReadDone(readRecord);
      }
      else if (len < sizeof(wt_logRecord))
            signal wtLogInterface.ReadComplete();
      else
            signal wtLogInterface.LogError();
}
```

La gestione dell'evento `readDone` può dare luogo a tre situazioni differenti: se l'operazione di lettura si è conclusa con successo ed il numero di byte letti dalla memoria flash è pari alla dimensione di un record, una copia dei dati estratti è inviata al modulo `wtServer;` in caso contrario, se il numero di byte letti è minore della dimensione di un record, allora non sono presenti ulteriori dati salvati, quindi viene segnalato il completamento delle operazioni di lettura; se si è verificato un errore, viene segnalata la condizione anomala.

Nota
I record salvati nella memoria flash sono letti uno alla volta.

A fronte della segnalazione di un evento `ReadDone`, il modulo `wtServer` si attiva per inviare al computer i dati estratti dal log:

```
event void Log.ReadDone(wt_logRecord *record){
      status = SENDING;
      r = (wt_logRecord*)call Packet.
                  getPayload(&msgToStation, sizeof(wt_logRecord));
      if (r == NULL){
            signal Log.LogError();
            return;
```

```
        }
        memcpy (r, record, sizeof(wt_logRecord));

        if (call AMSend.send(65535, &msgToStation, sizeof(wt_logRecord))
        != SUCCESS)
              signal wtServerInterface.EventTriggered(wtSrv_SENTABORTED,
        NULL);
}
```

Il parametro dell'evento `Log.ReadDone` rappresenta il record letto dalla memoria flash.
Nella porzione di codice sopra riportata si può osservare la chiamata al metodo `getPayload()`, definito nell'interfaccia `Packet` delle librerie di TinyOS, che si occupa di restituire il riferimento alla porzione di *payload* del pacchetto passatogli come parametro; nello specifico il pacchetto è quello contenuto nella variabile `msgToStation`.
Una volta che è stato ottenuto il riferimento `r` al payload del messaggio, il contenuto del `record` (letto dal componente `Log` dalla memoria flash) è copiato al suo interno con una `memcpy`.
Conclusa la preparazione del pacchetto, viene invocato il metodo `AMSend.send()` per l'invio di messaggi al computer attraverso il cavo USB.

Il componente `AMSend` è collegato (in `AppConfig.nc`) al componente di TinyOS `SerialActiveMessageC` che si occupa della gestione della comunicazione a basso livello sulla porta USB del sistema.
La libreria di TinyOS fornisce anche un componente chiamato `CC2420ActiveMessageC` che, utilizzato al posto del `SerialActiveMessageC`, invia i messaggi sfruttando il sistema di comunicazione radio 802.15.4 già presente sul Tmote; tale componente non è stato testato durante la fase di sviluppo della centralina,

La chiamata al metodo `send()` scatenerà, al termine della sua esecuzione, un evento `sendDone` che il modulo `wtServer` gestisce in questo modo:

```
event void AMSend.sendDone(message_t *msg, error_t error){
       free(r);
       if (status == IMMEDIATE) {
             status = WAITING;
             return;
       }
       status = WAITING;
       if (error == SUCCESS) {
             // try to read again
             call Timer.startOneShot(FLUSH_PERIOD);
       }
       else
             signal
                   wtServerInterface.EventTriggered(wtSrv_SENTABORTED,
                                                    NULL);
}
```

Se lo stato è `WAITING` viene fatto partire un timer che dopo `FLUSH_PERIOD` (5 ms) avvierà una nuova operazione di lettura dalla flash card.

```
	event void Timer.fired(){
		call Log.StartRead();
	}
```

In caso di errore durante la lettura dei log, il sistema ritorna in uno stato di attesa.

```
	event void Log.LogError(){
		status = WAITING;
	}
```

La segnalazione del termine delle operazioni di lettura è inoltrata al modulo `wtCore`...

```
event void Log.ReadComplete(){
	 signal wtServerInterface.EventTriggered(wtSrv_LOGSENT, NULL);  }
```

che gestisce l'evento `wtSrv_LOGSENT` spegnendo i led, avviando il timer che controlla le operazioni di sensing e cancellando il contenuto della memoria flash, rendendola disponibile per usi futuri.
In caso di errori durante l'invio dei log, al modulo `wtCore` viene notificato un evento di tipo `wtSrv_SENTABORTED`: in questo caso i log non vengono cancellati, in modo che sia possibile spedirli in un momento successivo, e viene acceso il led rosso (led 0) per segnalare l'errore di comunicazione.

```
event void wtServerInterface.EventTriggered
	(uint16_t eventType, wt_syncRecord * record){

	if (eventType == wtSrv_MSGRECEIVED){
		[...]
	}
	else if (eventType == wtSrv_LOGSENT || eventType == wtSrv_SENTABORTED){
		call Leds.led2Off();
		call Leds.led1Off();

		//The ball starts rolling
		if (call SenseTimer.isRunning() == FALSE)
			call SenseTimer.startPeriodic(SENSING_PERIOD);

		if (eventType == wtSrv_SENTABORTED)
			call Leds.led0On();

		else{
			call wtLogInterface.DropLog();
			call Leds.led0Off();
		}
	}
}
```

A questo punto, dopo aver avviato il `SenseTimer`, la centralina comincia ad interrogare periodicamente i sensori.

Interrogazione dei sensori

Segue la lista delle interfacce utilizzate dal modulo `wtSensors`: ognuna corrisponde ad uno dei sensori; inoltre è presente un'interfaccia (`PowerSwitch`) per il piedino che gestisce l'accensione e lo spegnimento della basetta.

```
module wtSensors{
      uses {
            interface GeneralIO as PowerSwitch;

            //digitali
            interface wtRainIf       as Rain;
            interface wtWindSpeedIf as WindSpeed;

            //analogici interni
            interface Read<uint16_t> as LightRead;
            interface Read<uint16_t> as InfraRead;
            interface Read<uint16_t> as TemperatureRead;
            interface Read<uint16_t> as HumidityRead;
            interface Read<uint16_t> as VoltageRead;

            //analogici esterni
            interface Read<uint16_t> as WindDirectionRea
            interface ReadStream<uint16_t> as PressureRead;
            interface ReadStream<uint16_t> as ExternalTemperatureRead;
      }
}
```

I metodi che espone a `wtCore` sono:

```
      command error_t init  ();
      command error_t sense (wt_logData * data);
```

Nel paragrafo precedente abbiamo visto come, in seguito alla ricezione di un messaggio di sincronizzazione, viene attivato per la prima volta il `SenseTimer` il cui compito è quello di scandire l'interrogazione dei sensori e mantenere operativa la centralina; l'*event handler* che si attiva ogni volta che questo timer scade non fa altro che invocare il comando `sense` del modulo `wtSensors`.

```
      event void SenseTimer.fired() {
            call wtSensorsIf.sense(&data);
      }
```

Il parametro `* data` rappresenta l'area di memoria in cui sono memorizzate le letture dei sensori e sarà popolato nel corpo di questa funzione.

```
command error_t wtSensorsIf.sense(wt_logData * data){
      if (data == NULL){
            signal wtSensorsIf.senseDone (NULL, FAIL);
```

```
            return FAIL;
      }
      else {
            sense = data;
            counter = 0;

            clean(extTempBuffer,  BUFFER_SIZE);
            clean(pressureBuffer, BUFFER_SIZE);

            //legge i sensori digitali
            sense->wind = call WindSpeed.get();
            sense->rain = call Rain.get();

            //legge i sensori interni
            call LightRead.read();
            call InfraRead.read();
            call TemperatureRead.read();
            call HumidityRead.read();
            call VoltageRead.read();

            //legge i sensori esterni
            call ExternalTemperatureRead.postBuffer(extTempBuffer,
                                                    BUFFER_SIZE);
            call ExternalTemperatureRead.read(SAMPLE_PERIOD);
      call PressureRead.postBuffer(pressureBuffer, BUFFER_SIZE);
            call PressureRead.read(SAMPLE_PERIOD);
            call WindDirectionRead.read();

            return SUCCESS;
      }
}
```

Di seguito la struttura della variabile `sense`:

```
      typedef struct wt_logData {
          uint16_t light;
          uint16_t infra;
          uint16_t temperature;
          uint16_t humidity;
          uint16_t voltage;
          uint16_t pressure;
          uint16_t ext_temperature;
          uint16_t wind;
          uint16_t direction;
          uint16_t rain;
      } wt_logData;
```

Dopo una verifica del parametro `data`, il metodo `sense` si occupa dell'inizializzazione dei buffer che conterranno le letture provenienti dal barometro e dal termometro di precisione: la procedura `clean` non fa altro che riempire questi array con degli zeri.
Segue la lettura dei contatori associati ai sensori digitali; i componenti che si occupano della gestione di tali sensori sono inizializzati nella fase di boot della centralina, mentre

l'aggiornamento dei contatori dipende dai segnali di interrupt nel caso dell'anemometro e dal polling del sensore nel caso del pluviometro, pertanto è asincrono rispetto all'esecuzione di `sense`.

Ad ogni ciclo di polling del pluviometro, se il valore che viene letto è un 1 significa che il sensore ha segnalato pioggia e quindi il componente `wtRain` incrementa il valore della variabile `rain`, un contatore interno.

```
event void RainTimer.fired(){
      if (rainStatus != call Rain.get()){
            rainStatus = !rainStatus;
            atomic rain += rainStatus;
      }
}
```

Il componente `wtSensors` chiama il metodo `get` dell'interfaccia `Rain` per leggere il contatore e azzerarlo: a questo punto nel campo `rain` della struttura `sense` sono contenuti gli scatti del pluviometro negli ultimi 5 secondi (`SENSING_PERIOD`).

```
command int wtRainIf.get(){
      atomic {
            int r = rain;
            rain = 0;
            return r;
      }
}
```

Il componente che gestisce l'anemometro funziona in maniera analoga, ma l'incremento del contatore interno non dipende dal polling del sensore, bensì dai segnali di interrupt provenienti dall'anemometro stesso.

```
async event void GpioInterrupt.fired(){
      atomic turns++;
      call Leds.led1Toggle();
}
command int wtWindSpeedIf.get(){
      atomic{
            int r = turns;
            turns = 0;
            return r;
      }
}
```

La chiamata a `led1Toggle` è stata inserita per motivi di debug. Da notare, nei corpi delle funzioni, la presenza di blocchi `atomic` per garantire l'assenza di *race conditions* sui valori dei contatori.

Dopo aver interrogato i sensori digitali, il metodo `sense` passa alla lettura dei sensori montati sul Tmote: tali sensori utilizzano i componenti forniti dalla libreria di TinyOS.

```
        new SensirionSht11C()      as HT,
        new HamamatsuS1087ParC()   as Light,
        new HamamatsuS10871TsrC()  as Infra,
        new VoltageC(),
```

`SensirionSht11C` è il modulo che virtualizza sia il termometro (`HT.Temperature`) che l'igrometro (`HT.Humidity`); entrambi espongono l'interfaccia `Read`, che permette l'esecuzione del comando `read` per la lettura del sensore a cui segue la segnalazione di un evento `readDone` per indicare la fine delle operazioni. Segue l'*event handler* per l'igrometro.

```
call HumidityRead.read();

event void HumidityRead.readDone(error_t result, uint16_t val){
      sense->humidity = (result == SUCCESS ? val : SENSORS_ERROR);
      up();
}
```

Un sistema analogo al precedente, con chiamate a `read` seguite da un *event handler* per l'evento `readDone` è stato utilizzato per gestire l'interrogazione di tutti i sensori analogici montati sul Tmote.
L'ultima istruzione dello handler è una chiamata al metodo `up`: si tratta dell'implementazione di una struttura di controllo di flusso semaforica, infatti la lettura dei sensori non è un'operazione sincrona, quindi per poter segnalare correttamente la fine di tutte operazioni è necessario un controllo su ciascun `readDone`.
Il metodo `up` genererà un evento `senseDone` solamente quando tutti i sensori avranno indicato chiamato il metodo, ovvero al termine di tutte le operazioni di lettura.

```
      void up () {
            atomic {
                  if (++counter == SENSORS_NUMBER)
                        signal wtSensorsIf.senseDone(sense, SUCCESS);
            }
      }
```

Il metodo `sense` termina con la lettura dei sensori analogici montati sulla basetta esterna: termometro di precisione, barometro e banderuola.

A questo scopo sono tati definiti i componenti `wtPressureC`, `wtExternalTemperatureC` e `wtWindDirectionC` che implementano l'inferfaccia `AdcConfigure`; tale interfaccia permette di definire una configurazione personalizzata degli ADC, per esempio specificando quale "canale" campionare e con che tensione di riferimento (vedi TEP 101).
La lettura della banderuola fa uso del componente `AdcReadClientC` che espone un'interfaccia `read` e genera un evento `readDone` al termine delle operazioni esattamente come accade per i sensori ospitati sul Tmote.
L'interrogazione del termometro di precisione e del barometro, invece, sfrutta il componente `AdcReadStreamClientC`: si tratta di un componente presente nella libreria di TinyOS che permette di acquisire un *stream* di dati da un sensore. Il metodo `sense` di `wtSensors` medierà queste letture in modo da limitare gli errori dovuti al rumore.

La lettura di uno stream di dati è un processo in due passi: c'è una fase preparatoria (`postBuffer`) in cui viene indicato l'array che conterrà i risultati a cui segue l'acquisizione vera e propria (`read`) dei dati; il parametro SAMPLE_PERIOD indica il periodo di tempo, espresso in µs, che intercorre tra due campionamenti successivi.

```
call PressureRead.postBuffer(pressureBuffer, BUFFER_SIZE);
call PressureRead.read(SAMPLE_PERIOD);
```

Quando il buffer è stato riempito viene generato un evento `bufferDone` che sarà gestito dal modulo `wtSensors`.

```
event void PressureRead.bufferDone(error_t result,
                                   uint16_t *buf, uint16_t count){
      if (result != SUCCESS){
            sense->pressure = SENSORS_ERROR;
      }
      else {
            sense-> pressure = 0;
            for (i_press = 0; i_press < count; i_press++){
                  sense->pressure += buf[i_press];
            }
            sense->pressure /= count;
      }
      up();
}
```

In questo caso (del tutto simile a quanto avviene per il termometro di precisione) insieme alla segnalazione dell'evento vengono restituiti `*buf`, un puntatore all'array che mantiene i dati in memoria e `count` che ne rappresenta la lunghezza. Il metodo `sense` può quindi calcolare la media delle letture e salvarla in un campo della struttura `sense`.

```
event void WindDirectionRead.readDone(error_t result, uint16_t val){
      sense->direction = (result == SUCCESS ? findDirection(val)
                                                : SENSORS_ERROR);
            up();
}
```

La lettura della banderuola è del tutto analoga a quella dei sensori analogici ospitati sul Tmote, ma dopo la conversione da un valore di tensione ad un numero a 12 bit è presente un'ulteriore elaborazione dei dati per mappare, **già all'interno della centralina**, il valore letto dal sensore in una direzione in gradi rispetto alla direzione Nord. Il mapping è effettuato dalla funzione `findDirection`...

```
uint16_t findDirection (uint16_t ADCCount){
      int il = 0;
      int ih = 15;
      while (ih - il > 1) {
            int mid = (il + ih)/2;
            if (ADCCount == compass_rose[mid][0])
```

```
                return compass_rose[mid][1];
            else if (ADCCount < compass_rose[mid][0]){
                ih = mid;
            }
            else {
                il = mid;
            }
      }
      //se esce qui senza ritornare, allora indexh = indexl + 1
      return (ADCCount - compass_rose[il][0] < compass_rose[ih][0] -
ADCCount) ? compass_rose[il][1] : compass_rose[ih][1];
}
```

che esegue la ricerca binaria del valore argomento all'interno della tabella `compass_rose` definita nel file `settings.h` e riportata qui di seguito.
Il primo elemento di ogni coppia è il valore numerico corrispondente al valore di tensione generato dal sensore quando la paletta indica una delle sedici direzioni della rosa dei venti; il secondo è una costante enumerativa associata alla direzione.

```
uint16_t const compass_rose [][2]= {{262,   6},  //ESE
                                    {336,   4},  //ENE
                                    {369,   5},  //E
                                    {508,   8},  //SSE
                                    {737,   7},  //SE
                                    {975,  10},  //SSW
                                    {1147,  9},  //S
                                    {1622,  2},  //NNE
                                    {1843,  3},  //NE
                                    {2400, 12},  //WSW
                                    {2523, 11},  //SW
                                    {2809, 16},  //NNW
                                    {3145,  1},  //N
                                    {3309, 14},  //WNW
                                    {3784, 13},  //W
                                    {3914, 15}   //NW
                                   };
```

Quando tutti gli *envent handler* hanno popolato la struttura `sense` con le letture e hanno invocato il metodo `up`, il meccanismo di *lock* semaforico scatena l'evento `senseDone`, gestito dal modulo `wtCore`.

```
event void wtSensorsIf.senseDone(wt_logData *d, error_t error){
      if (error != SUCCESS) {
            call Leds.led2On();
      }
      else {
            uint64_t now = call SenseTimer.getNow() + lastTimeStamp - lastMsgArrTime;
            call wtServerInterface.SendImmediate (data, now);

            if (call SenseTimer.getNow() - lastMsgArrTime > SERVER_TIMEOUT){
                  call wtLogInterface.Append (data, now);
                  call Leds.led1On();
```

```
            }
        }
}
```

Il componente `wtCore` calcola il timestamp da allegare al record delle letture dei sensori; risulta necessaria la correzione del timestamp con l'ultimo orario ricevuto dal server per evitare l'eccessiva deriva del clock del Tmote rispetto a quello del computer.
In seguito viene chiamato il metodo

```
call wtServerInterface.SendImmediate (data, now);
```

che inviare alla LogStation il pacchetto con timestamp e dati meteorologici.
Il controllo successivo verifica la "presenza" del computer: se non è stato ricevuto un timestamp da più di `SERVER_TIMEOUT` (45 secondi) si presume che il server non sia in grado di ricevere i dati, quindi si invoca il salvataggio del record nella memoria flash; contestualmente viene acceso il led 1 per notificare l'assenza di comunicazione.

```
call wtLogInterface.Append (data, now);
call Leds.led1On();
```

Comunicazione con il server

Le comunicazioni tra la centralina meteo ed il PC prevedono lo scambio di due tipi diversi di messaggio: il primo è un messaggio di sincronizzazione, inviato periodicamente dal server al Tmote, contenente come unico campo un intero a 64 bit che rappresenta il timestamp del server; il secondo è un messaggio, inviato dal Tmote al server, che contiene le rilevazioni.
La dimensione del payload di tutti i pacchetti è quella di default, cioè 28 byte (cfr. TEP 111).
Lo scambio dei dati "a basso livello" avviene sulla porta USB del Tmote.

Server 2 Mote

A livello del server, i messaggi sono visti come degli oggetti che contengono al loro interno un array di byte - il payload - la cui dimensione è definita dalla costante `PKT_SIZE` in `MoteMessage.java` ed è pari al valore di default specificato da TinyOS (cfr. TEP 111).

```
public MoteMessage () {
      super(new byte[PKT_SIZE]);
}
```
[3]

È il MoteConnector che, con un intervallo periodico definito dalla costante `KEEPALIVE`, crea ed invia i messaggi di sincronizzazione

```
Message msg = new
Message(this.listener.toByteArr(System.currentTimeMillis()));
moteIf.send(0, msg);
```
[4]

[3] il costruttore dei messaggi, alla riga 90 nel file MoteMessage.java
[4] la costruzione e l'invio di un messaggio di sincronizzazione, alla riga 45 di MoteConnector.java

Il Tmote interpreterà il payload come una struttura di tipo `wt_syncRecord`; i byte in ecesso oltre agli 8 necessari per rappresentare il timestamp sono considerati padding e vengono scartati dall'applicazione.

```
typedef struct wt_syncRecord {
    uint64_t timestamp;
} wt_syncRecord;
```

Mote 2 Server

Il Tmote definisce il payload dei messaggi come una struttura dati che contiene il timestamp relativo all'ora in cui è stata eseguita la lettura ed un record che contiene i valori acquisiti dai sensori; anche in questo caso la lunghezza è pari a quella di default.

```
typedef struct wt_logRecord {
    uint64_t   datetime;
    wt_logData data;
} wt_logRecord;
```

La creazione e l'invio dei messaggi sono affidate al modulo `wtServer` che se ne occupa implementando il comando `SendImmediate`.
`msgToStation` è la struttura dati di tipo `message_t` (vedi TEP 111) che sarà inviata al server: per prima cosa `SendImmediate` cerca di ottenere un puntatore all'area corrispondente al payload, in caso di successo crea una nuova struttura di tipo `wt_logRecord` e la copia all'interno del messaggio, poi, con l'esecuzione del comando `AMSend.send` (vedi TEP 113 e TEP 116), spedisce il messaggio all'indirizzo 65535 (broadcast).

```
command error_t wtServerInterface.SendImmediate(wt_logData data, uint64_t time){
  status = IMMEDIATE;

   r = (wt_logRecord*) call Packet.getPayload(&msgToStation, sizeof(wt_logRecord));
   if (r == NULL) {
    signal Log.LogError();
    return FAIL;
   }

   rimmediate = wt_createLogRecord(data, time);
   memcpy (r, rimmediate, sizeof(wt_logRecord));

   if (call AMSend.send(65535, &msgToStation, sizeof(wt_logRecord)) != SUCCESS)
     signal wtServerInterface.EventTriggered(wtSrv_SENTABORTED, NULL);

   free(rimmediate);
   return SUCCESS;
}
```
[5]

Un procedimento analogo è eseguito quando il Tmote spedisce un record proveniente dal log; in questo caso non è necessario creare al momento il messaggio, bensì vengono inviati i dati recuperati dalla memoria flash.

```
event void Log.ReadDone(wt_logRecord *record){
       status = SENDING;
```

[5] wtServer.nc, riga 99

```
        r = (wt_logRecord*)call Packet.getPayload(&msgToStation,
sizeof(wt_logRecord));
        if (r == NULL){
              signal Log.LogError();
              return;
        }

        memcpy (r, record, sizeof(wt_logRecord));

        if (call AMSend.send(65535, &msgToStation, sizeof(wt_logRecord)) != SUCCESS)
              signal wtServerInterface.EventTriggered(wtSrv_SENTABORTED, NULL);
    }
```
[6]

Lato server questi messaggi sono incapsulati in oggetti di tipo MoteMessage, di fatto dei wrapper per gli array di byte che compongono i payload. La classe MoteMessage mette a disposizione una serie di costanti e di metodi per estrarre i campi dai messaggi.

```
    //posizione dei campi nel pacchetto
    private static final int POS_TIME  = 0;
    private static final int POS_LIGHT = 8;
    private static final int POS_INFRARED = 10;
    private static final int POS_TEMPERATURE = 12;
    private static final int POS_HUMIDITY = 14;
    private static final int POS_VREF = 16;
    private static final int POS_PRESSURE = 18;
    private static final int POS_EXT_TEMPERATURE = 20;
    private static final int POS_WIND = 22;
    private static final int POS_DIRECTION = 24;
    private static final int POS_RAIN = 26;

    //estrae un campo di tipo long
    private long getLong (int pos) {
       try {
             byte[] arr = Arrays.copyOfRange(super.dataGet(), pos, pos + 8);
             ByteArrayInputStream bais = new ByteArrayInputStream(arr);
             DataInputStream dis = new DataInputStream(bais);
             return Long.reverseBytes(dis.readLong());
       } catch (Exception e) { }
       return -1;
    }

    //estrae un intero
    private int getInt (int pos) {
       int r = 0;
       byte data[] = super.dataGet();
       for (int i = pos; i < pos + 2; i++) {
             r |= (data[i] & 0xFF) << 8*(i-pos);
       }
       return r;
    }
```
[7]

[6] wtServer.nc, riga 51
[7] MoteMessage.java, riga 10

Protocollo di comunicazione

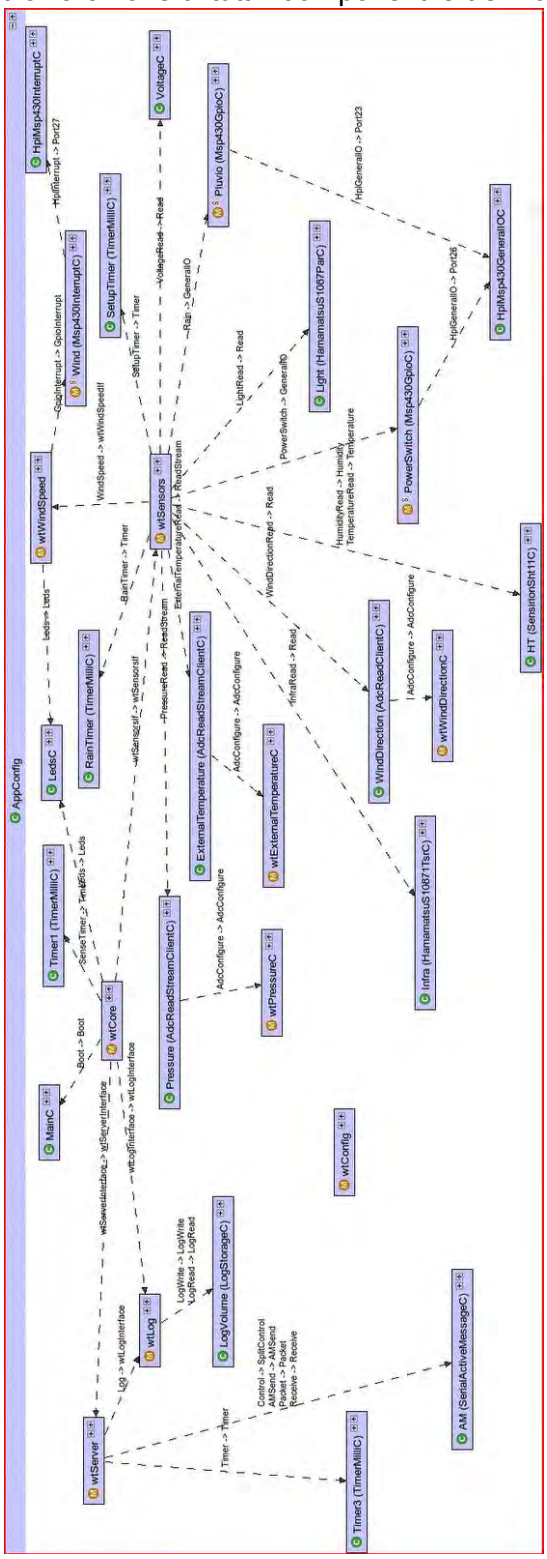
I messaggi generati dal Tmote sono inviati al server sulla porta USB del dispositivo adottando una strategia **push**(cioè il server non richiede i dati alla periferica, ma è la periferica che, quando esegue un ciclo di lettura dei sensori o legge le rilevazioni salvate sulla memoria flash, invia i dati al server), **best effort** e **unacknowledged**.
La scelta di non predisporre un sistema di ack e ritrasmissione dei dati è stata dettata principalmente dalla presenza di una connessione *wired* (quindi ritenuta affidabile) tra il Tmote ed il server e dalla considerazione che la perdita di "qualche" messaggio non è poi così grave, perché le rilevazioni puntuali sono tante (una ogni 5 secondi) mentre agli utenti sono mostrati dei valori aggregati. Un protocollo così semplice, inoltre, è di facile implementazione e non genera overhead dovuti alla gestione del meccanismo di ack e ritrasmissione.

Anche i pacchetti di sincronizzazione inviati dal server al Tmote sono inviati con una politica **push**, **best effort** e **unacknowledged**; in questo caso, però, la perdita di uno di questi messaggi è considerata una condizione d'errore:

- prima della ricezione del primo timestamp, il Tmote non leggerà i sensori poiché sarebbe impossibile associare una data ed un orario ad ogni rilevazione
- dopo la ricezione del primo timestamp, il Tmote aspetta di ricevere un pacchetto di sincronizzazione ad intervalli regolari: se allo scadere di un timeout il messaggio non è arrivato, il Tmote assumerà che ci sia un errore lato server, quindi comincerà a loggare i dati sulla memoria flash
- dopo l'invio di ogni timestamp, il server dovrebbe ricevere i dati provenienti dalla periferica (una sorta di ack implicito): controllando la dimensione della tabella in cui vengono memorizzate temporaneamente le rilevazioni è possibile sapere se il Tmote sta effettivamente inviando i dati; in caso contrario il software in esecuzione sul server supporrà che ci sia qualche problema di comunicazione e si riavvierà

Componenti software

Nel grafico seguente è illustrata la struttura dell'applicazione sviluppata in NesC per il Tmote; la dichiarazione di tutti i componenti è definita all'interno del file `AppConfig.nc`.



File di header
Oltre ai sorgenti con estensione `.nc` sono presenti due file di header: `settings.h` e `StorageVolumes.h`.
Il primo contiene la definizione delle costanti (periodicità dei timer, numero dei sensori, tipi enumerativi, tabelle di look up) e delle strutture dati utilizzate:
`struct wt_logData` è la struttura utilizzata per memorizzare le letture dei sensori
`struct wt_logRecord` è un wrapper per `wt_logData` al quale aggiunge un riferimento temporale (`timestamp`)
`struct wt_syncRecord` è la struttura del pacchetto di sincronizzazione inviato periodicamente dal server
il secondo contiene una specie di tabella delle partizioni della memoria flash.

AppConfig.nc
Rappresenta l'entry point del sistema e definisce le connessioni tra le componenti. Il grafico precedentemente riportato è stato costruito sulla base delle dichiarazioni presenti in questo file.

wtCore.nc
È il nodo principale dell'applicazione e si occupa del bootstrap del dispositivo chiamando le procedure che inizializzano i sensori e avviano il modulo di ascolto dei messaggi del server. Questo componente è sensibile all'evento che segnala il completamento delle rilevazioni, la cui attivazione scatena una richiesta di invio dei dati per il modulo `wtServer` o, qualora sia trascorso un tempo superiore a `SERVER_TIMEOUT`[8] dalla ricezione dell'ultimo messaggio di sincronizzazione, una chiamata al modulo di logging `wtLog` per la memorizzazione dei dati sul supporto flash.

Led
Il modulo `wtCore` gestisce anche i segnali minimali di debug dell'applicazione mediante i tre led montati sul Tmote:
il led **rosso** (`led0`) indica un errore avvenuto in fase di comunicazione con il server (evento `wtSrv_SENTABORTED`)
il led **giallo/verde** (`led1`) segnala che il Tmote ha perso la connessione con il server e quindi ha attivato la procedura di logging dei dati sulla memoria flash
il led **blu** (`led2`) rimane acceso in caso di errori in lettura dai sensori, mentre lampeggia brevemente quando il Tmote riceve il pacchetto di sincronizzazione dal server e commuta (toggle) ogni volta che il dispositivo cattura un interrupt generato dall'anemometro.

wtLog.nc
È il *mediator* per le interfacce di *TinyOS* `LogRead` e `LogWrite`: gestisce le letture e scritture dei record sulla memoria flash del dispositivo.

wtServer.nc
Si tratta del componente che si occupa ad alto livello della comunicazione tra il Tmote e la stazione di processing e archiviazione dei dati; ad un livello più basso, lo scambio dei dati

[8] la soglia oltre la quale la connessione è considerata persa e il Tmote comincia a registrare i dati sulla memoria flash è definita dalla costante `SERVER_TIMEOUT` nel file `settings.h` ed è pari a 45 secondi; tale soglia **deve** essere maggiore dell'intervallo di tempo che intercorre tra due invii consecutivi del timestamp di sincronizzazione da parte del server.

sfrutta il componente `SerialActiveMessageC` di *TinyOs* che utilizza la porta USB montata sul dispositivo.
I messaggi in arrivo servono unicamente per la sincronizzazione del clock; la loro ricezione causa la segnalazione di un evento `wtSrv_MSGRECEIVED` a cui il modulo `wtCore` reagisce aggiornando il timestamp locale con quello presente nel messaggio e, se non è già attivo, avviando il timer per il polling dei sensori.
Il comando `SendImmediate` di questo modulo, invece, si occupa della creazione dei messaggi di tipo `wt_logRecord` e del loro invio **push** verso il server.

wtSensors.nc
È il modulo della stazione meteorologica che gestisce l'attivazione e la lettura di tutti i sensori, sia quelli installati a bordo del Tmote, sia quelli montati sulla basetta esterna. Produce un record di tipo `wt_logData`[9] di `uint16_t` (interi a 16 bit senza segno) che rappresentano l'output degli ADC collegati ai sensori analogici oppure il valore di un contatore nel caso dei sensori digitali. Le conversioni nelle rispettive unità di misura del Sistema Internazionale sono fatte a posteriori dalla `LogStation`[10].

Il comando `init` di `wtSensors` provvede all'accensione dei sensori ospitati sulla basetta[11] "esterna" (termometro di precisione, barometro, anemometro): il circuito che ne controlla l'alimentazione è comandato dal connettore `GIO3` del Tmote, che è gestito dal componente `Msp430GpioC` di *TinyOS*.
`Init` avvia i sensori invocando i comandi `makeOutput()` e `set()` di questo componente; lo stesso comando si occupa, inoltre, di impostare i piedini associati ai sensori digitali come pin di input, avviare i contatori a loro associati[12] e preparare i buffer per le misure della temperatura e della pressione atmosferica.

Il comando `sense` è chiamato ad intervalli regolari[13], scatena la lettura dei sensori, sia digitali che analogici, e popola con i risultati una struttura di tipo `wt_logData`. Le letture dei sensori analogici sono effettuate in parallelo mediante chiamate asincrone a funzioni, di conseguenza, per segnalare il completamento delle operazioni è stato implementato un semaforo: l'evento `senseDone`, che segnala il termine di un ciclo di sensing è lanciato solamente quando tutti i sensori hanno "abbassato" il semaforo.[14]

Attenzione

---

[9] la struttura di wt_logData è definita all'interno del file di header `settings.h`

[10] vedere i metodi della classe `Conversion.java`

[11] la basetta integra un led che si accende per segnalare che è alimentata; nel caso in cui si volesse rendere più efficiente il sistema dal punto di vista del consumo energetico (per esempio perché si vuole alimentare la stazione meteo a batterie) è possibile spegnere i sensori esterni con il comando `clr()` di `Msp430GpioC`; in questo caso è bene predisporre anche un *intervallo di setup* necessario agli stessi sensori per entrare a regime dopo l'accensione

[12] eseguendo `Rain.init()` e `windSpeed.init()`

[13] l'intervallo di sensing è definito dalla costante `SENSING_PERIOD` nel file `settings.h`

[14] in realtà la primitiva che agisce sul semaforo si chiama `up()`; in una versione precedente del programma esisteva anche una primitiva `down()` che veniva invocata al termine delle letture dei sensori posizionati sulla basetta esterna e che serviva a segnalare al modulo `wtSensors` che era possibile spegnerli

Nella definizione della struttura `wt_logData` nel file `settings.h` compaiono due campi per la temperatura, `temperature` ed `ext_temperature`: il primo corrisponde alla temperatura misurata dal termometro / igrometro presente sul Tmote, il secondo alla temperatura letta dal termometro di precisione montato sulla basetta esterna.
Lato server (Java, database, web) abbiamo invece `temperature` e `temperature_int`: la prima si riferisce ai valori del termometro di precisione, mentre la seconda alle letture del termometro / igrometro integrato.
Il *conflitto* sui nomi è dovuto al fatto che la parte di processing, archiviazione e pubblicazione dei dati che risiede sul server è stata sviluppata successivamente alla parte di acquisizione delle misure che è in esecuzione sul Tmote; lato server, solo una delle due misure di temperatura - quella proveniente dal termometro di precisione - è mostrata all'utente (e quindi, per brevità, è stata denominata `temperature`), mentre l'altra è utilizzata unicamente per la correzione del tasso di umidità.

Sempre nella struttura `wt_logData` compare un campo `voltage` che rappresenta il valore della tensione di alimentazione del dispositivo: tale valore viene letto dal Tmote, ma non viene ulteriormente elaborato dal server; potrebbe essere utile nel caso la periferica fosse alimentata a batterie, per misurare il livello di carica.
Sono inoltre presenti due campi per i valori della radiazione visibile ed infrarossa (campi `light` e `infra`). Tali valori non sono mostrati all'utente che visita il sito internet poiché la stazione meteorologica *attualmente* non dispone di una finestra per il funzionamento dei fotodiodi.

Sensori analogici sul Tmote
Per la lettura dei sensori installati sul Tmote si è ricorso a componenti già presenti in *TinyOS*:
`SensirionSht11C` gestisce il sensore di umidità e temperatura
`HamamatsuS1087ParC` rappresenta il sensore di luce totale
`HamamatsuS10871TsrC` rappresenta il sensore di luce infrarossa
Le operazioni di lettura sfruttano l'interfaccia *`Read`* implementata da questi componenti.

Sensori analogici sulla basetta esterna
La banderuola per la misura della direzione del vento è "virtualizzata" dal componente `wtWindDirectionC` che fornisce tutti i parametri (numero di canale, tensioni di riferimento, ecc...) per la configurazione dell'ADC.
Anche in questo caso le operazioni di lettura sfruttano l'interfaccia *`Read`*, ma contrariamente agli altri sensori analogici, che producono un valore numerico che in un secondo momento deve essere convertito dal server, per quanto riguarda la banderuola il risultato è calcolato già all'interno del Tmote: nel campo `direction` di `wt_logData`, infatti, sarà inserito il valore numerico che rappresenta la direzione che più si avvicina alla rilevazione della paletta. Questo calcolo è eseguito dalla funzione `findDirection`[15] che mappa la lettura analogica in un numero tra 1 e 16 secondo la tabella `compass_rose`, definita nel file `settings.h`.

Il termometro di precisione ed il barometro sono "virtualizzati" rispettivamente dai moduli `wtExternalTemperatureC` e `wtPressureC`, che forniscono i parametri di configurazione degli ADC.
In questo caso, poiché il rumore sulle misure generava delle notevoli fluttuazioni dei valori pubblicati, per cercare di attenuarlo, ad ogni ciclo di sensing non viene eseguita una sola

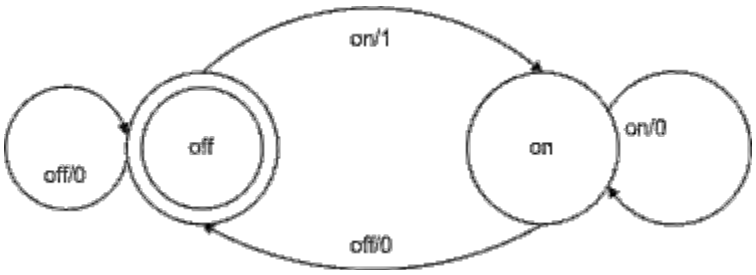
[15] `find_direction` implementa un algoritmo di ricerca binaria in una tabella ordinata

lettura, bensì viene riempito un buffer[16] di dati; al server è quindi inviato il valore medio delle rilevazioni.

Sensori digitali
La misura della velocità del vento è gestita in maniera differente perché è comandata dai segnali di *interrupt* generati dalla rotazione delle palette dell'anemometro: il Tmote conta gli impulsi ricevuti (fronte di salita) durante l'intervallo di sensing e li inserisce nel campo `wind` della struttura `wt_logData`. Questo numero, che corrisponde al numero di rotazioni delle palette, sarà successivamente diviso per l'intervallo di tempo per ottenere i giri al minuto e da questi la velocità del vento espressa in chilometri orari; la conversione è effettuata dalla `LogStation`.

Attenzione
Spesso il contatore segnala che non ci sono stati interrupt per periodi di tempo piuttosto lunghi (anche di qualche giorno), anche se le palette dell'anemometro continuano a girare: potrebbe trattarsi di un problema dovuto all'umidità che si forma all'interno della centralina oppure di un falso contatto tra il filo dell'anemometro ed il piedino di interrupt del Tmote; per identificare la causa del problema, ad ogni ricezione del segnale di interrupt il led blu (`led2`) commuta.

Il pluviometro, collegato alla porta `GIO4` gestita dal modulo `HplMsp430GeneralIOC` di *TinyOS*, merita un discorso a parte.
Il sensore funziona come un interruttore che si chiude ogni volta che raccoglie una sufficiente quantità d'acqua, ma data l'assenza di un'altra porta su cui ascoltare segnali di interrupt (e le complicazioni legate alla gestione di interrupt potenzialmente *nested* provenienti da fonti differenti) è stato collegato ad un condensatore che ne memorizza l'impulso per un tempo sufficiente ad effettuarne la lettura. La gestione software di questo sensore è stata realizzata mediante un automa a stati finiti.



Nella figura è rappresentato il grafico di Mealy implementato in lettura al livello del segnale presente sulla porta di input: `on` rappresenta livello alto, `off` il livello basso; gli input sono dati da polling sulla porta e gli output vengono sommati in un contatore che allo scadere del tempo di sensing viene memorizzato nel record di lettura; l'intervallo di tempo tra due letture successive del sensore è definito dalla costante `POLLING_PERIOD`.

[16] la dimensione dei buffer delle letture e la durata del periodo di campionamento sono definite dalle costanti `BUFFER_SIZE` e `SAMPLE_PERIOD` nel file `settings.h`; i valori derivano da prove empiriche eseguite in laboratorio: con combinazioni differenti si ottengono troppo pochi dati da mediare (e quindi un maggiore rumore) oppure si verifica il blocco del Tmote

Cross-compilazione e installazione del firmware

In fase di sviluppo è possibile sfruttare le funzionalità del plugin Yeti2 per compilare e caricare il programma sul Tmote.
Successivamente, per esempio quando la stazione meteorologica è già stata collegata al server meteo ed è necessario rilasciare una nuova versione dell'applicazione, è possibile compilare il codice sul proprio computer (con Eclipse e Yeti2), caricarlo sul server meteo (per esempio con `scp`) e flashare il Tmote digitando

```
tos-bsl --comm-port=PORTA -I --telosb -e -p -v -r HEX_FILE
```

in una shell `ssh`, indicando la porta a cui è collegato il Tmote ed il file binario che si desidera installare, così come viene prodotto dal compilatore.
La lista completa delle opzioni di `tos-bsl` ed il loro significato è disponibile digitando

```
tos-bsl --help
```

nel terminale; alcune guide alla cross compilazione sono disponibili online

Programmazione del Server

Applicazione Java (LogStation)

I sorgenti dell'applicativo Java si trovano in `~/InstallFiles/source/LogStation`, mentre i file `.class` e la copia del file di configurazione letta all'avvio del programma sono in `~/InstallFiles/source/bin`.

Avvio e terminazione del servizio / programma

Il sistema è configurato per avviare automaticamente la Logstation come servizio della macchina secondo quanto specificato nel file `/etc/init.d/weather`.

Per avviare manualmente il servizio, digitare nel terminale

```
sudo /ect/init.d/weather start
```

mentre per terminarne l'esecuzione scrivere

```
sudo /ect/init.d/weather stop
```

Attenzione
L'avvio e la terminazione del servizio *weather* richiede i privilegi di superuser: il sistema richiederà l'inserimento della password prima di procedere con l'operazione richiesta.

Se monit è in funzione sul server ed è opportunamente configurato[17], terminare l'esecuzione di *weather* causerà la reazione dello strumento di monitoring che, rilevata una condizione d'errore, provvederà ad avviare una nuova LogStation. Se dovesse essere necessario sospendere l'esecuzione del servizio, per esempio per rilasciare una patch, è necessario interrompere monit prima di procedere con l'operazione.

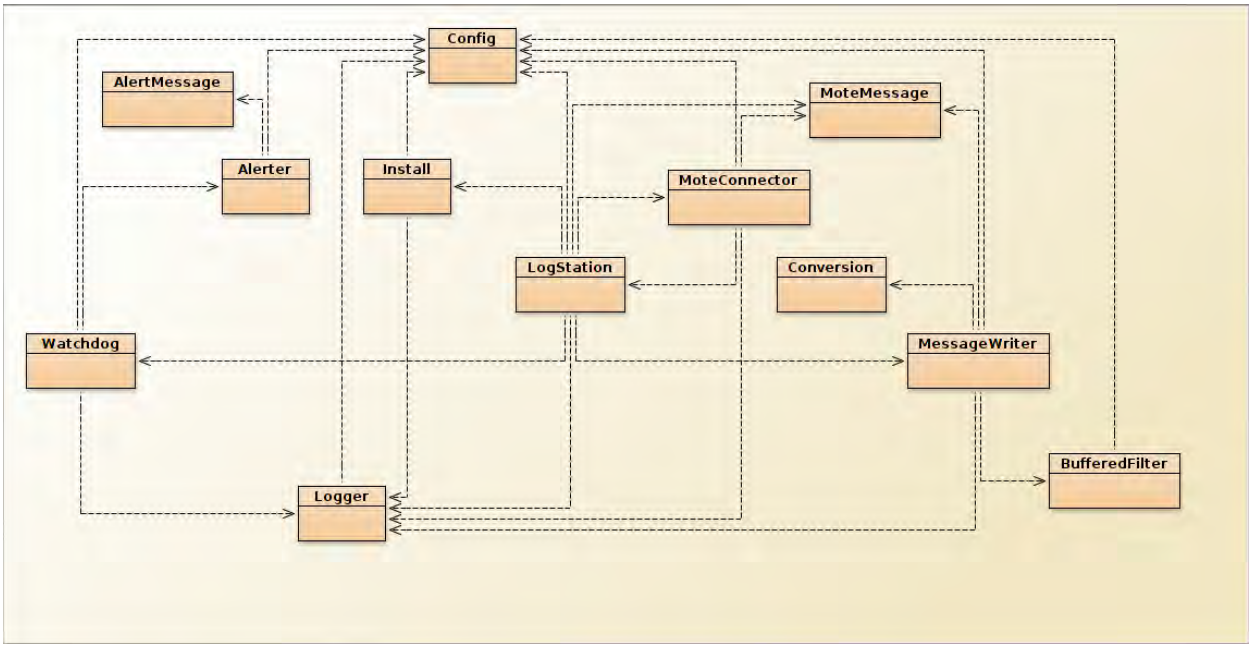
Per lanciare manualmente l'applicazione, spostarsi nella directory che contiene i file compilati e digitare

```
java LogStation
```

eventualmente fornendo come argomento a riga di comando la porta a cui è collegato il Tmote: non appena comincerà lo scambio di dati, compariranno a video i valori letti dagli ADC in formato human-readable. In questo caso un semplice `CTRL+C` è sufficiente per terminare l'esecuzione del programma.

[17] Per maggiori dettagli sulla configurazione di Monit leggere la sezione "Strumenti di monitoring" di questo documento.

Struttura dell'applicazione



L'applicazione Java è composta da due parti: una dedicata alla ricezione, processing e archiviazione dei dati meteorologici rilevati dal Tmote; l'altra si occupa del monitoring del database, della segnalazione degli errori e del logging.

Componenti comuni

Config.java
Fornisce i metodi utilizzati dagli altri componenti dell'applicazione per leggere i parametri specificati nel file di configurazione; in caso di errore (file di configurazione non trovato, permessi insufficienti, formato errato delle costanti numeriche) il programma terminerà con un codice di errore come indicato nel file README.txt allegato ai sorgenti.

Logger.java
Fornisce i metodi per il logging dei messaggi di errore sul file `/var/log/logstation.log`; le operazioni di scrittura possono fallire se il programma viene avviato da un utente che non gode di permessi sufficienti per manipolare tale file.
Il verificarsi di errori in fase di logging non interrompe il funzionamento dell'applicazione; è tuttavia possibile modificare la riga 43 del sorgente per forzare la terminazione del programma anche in questo caso.

Rilevazione, processing e archiviazione

MoteMessage.java
Definisce le costanti ed i metodi per il *parsing* (soluzione dei problemi di endianess, estrazione dei campi e loro conversione da esadecimale a decimale) dei pacchetti ricevuti dal Tmote.

BufferedFilter.java
Per cercare di attenuare il rumore che affligge le misure di pressione e temperatura, i valori letti dal Tmote sono conservati in una coda FIFO prima di essere inseriti nel database.
Alla ricezione di ogni nuovo dato, l'ultimo (il più vecchio) della coda viene scartato, quello nuovo è inserito in testa alla coda e la media delle ultime rilevazioni è scritta nella base di dati.
La dimensione dei buffer è letta dal file di configurazione.
Le code sono mantenute in memoria come elementi di una HashMap a cui si può accedere specificando il nome come chiave di ricerca.

Conversion.java
Definisce i metodi per la conversione dei dati binari rilevati dal Tmote in misure di grandezze fisiche espresse in unità del Sistema Internazionale.
Quasi tutte le costanti impiegate durante i calcoli sono definite all'interno di questa classe; fanno eccezione i fattori di correzione[18] per la temperatura e la pressione e l'offset[19] per la direzione del vento, che sono parametri specificati dall'utente all'interno del file di configurazione.
La procedura può fallire, sollevando un'eccezione, se il valore da convertire corrisponde alla costante `SENSORS_ERROR` specificata nel file di header `settings.h` dell'applicazione eseguita dal Tmote.

MessageWriter.java
Gestisce le operazioni di scrittura delle rilevazioni meteorologiche sul database.
In caso di errore (dovuto a problemi con la base di dati o a valori errati rilevati dai sensori), causa la scrittura di un record sul file di log.

Install.java
Contiene i metodi che creano la struttura della base di dati al primo avvio del programma.
Può generare errori se l'utente con cui l'applicazione si collega al database non gode di privilegi sufficienti[20]; aggiunge al file di log l'elenco delle tabelle e degli eventi creati con successo.

MoteConnector.java
Si tratta del *Thread* che invia periodicamente il segnale orario al Tmote; l'intervallo di tempo tra due invii successivi è indicato dal parametro `KEEPALIVE` del file di configurazione.
I metodi di questa classe fanno uso della libreria Java di *TinyOs* (net.tinyos.message.*, net.tinyos.util.*, net.tinyos.packet.*) per la generazione e l'invio di messaggi e aggiungono al file di log i record relativi all'inizio e alla fine della connessione tra il server ed il Tmote.

LogStation.java
È il componente principale dell'applicazione.
Ad ogni avvio, utilizza i metodi della classe *Install* per controllare l'esistenza di tutti gli oggetti della base di dati necessari per l'archiviazione delle rilevazioni, scrive il proprio PID in un file perché i tool di monitoring (monit) possano controllarlo, lancia un nuovo thread *MoteConnector* per inviare l'orario del server al Tmote e scatenare così il processo di lettura dei sensori, avvia le procedure di monitoring e si mette in ascolto dei messaggi provenienti dal Tmote; alla ricezione di ogni pacchetto, invoca i metodi di *MessageWriter* per scrivere le misure sul database.

---

[18] vedere l'appendice sulle formule di conversione

[19] idem

[20] si veda il paragrafo relativo alle utenze della base di dati

Monitoring

AlertMessage.java
Semplice classe che descrive un messaggio d'errore, corredato di data. È stata implementata seguendo il pattern dell'*oggetto immutabile*.

Alerter.java
Thread che si occupa del mantenimento della coda delle notifiche e del loro invio periodico via email.
Richiede l'esecuzione del programma *sendemail* per la spedizione della posta: se la LogStation venisse lanciata con permessi non sufficienti per eseguire l'invio, o per qualche ragione il programma non dovesse funzionare, non sono disponibili informazioni di debug; per attivarle è necessario modificare la riga 93 del codice sorgente.
Tutti i messaggi generati dalla stessa condizione d'errore sono mantenuti all'interno di uno stesso vettore; questi vettori sono organizzati in una HashMap che ha i nomi dei metodi che li hanno segnalati come chiavi di ricerca.
Ogni volta che viene creato un nuovo messaggio, il sistema controlla di non avere già una coda per lo stesso metodo che sta segnalando l'errore e di non aver già inviato una notifica analoga (relativa allo stesso metodo) negli ultimi `MSG_REPEAT` minuti; se così fosse, la segnalazione viene accodata, altrimenti è inviata immediatamente. Periodicamente un thread provvede a inviare per email tutte le notifiche non ancora spedite, a svuotare le code e ad aggiornare i timestamp relativi ai tempi di invio.
Al momento le email sono inviate agli indirizzi di posta elettronica degli sviluppatori, per cambiare i destinatari è necessario modificare la riga 84 del sorgente.
Il parametro `MSG_REPEAT` è presente nel file di configurazione della Logstation.

Watchdog.java
È il processo principale della parte di monitoring: controlla periodicamente che la dimensione della tabella in cui vengono scritti i dati non contenga meno (più) di un certo numero di record, che lo scheduler di MySQL sia in funzione e che sia presente il *PID file* della LogStation; se uno di questi controlli dovesse fallire, Watchdog scriverà un record nel file di log, tenterà di inviare una notifica via email e causerà il riavvio[21] dell'applicazione.
Prima del primo controllo, Watchdog rimane in *sleep* per un certo intervallo di tempo per non causare la terminazione anomala della LogStation a causa di "errori transienti" (per esempio un database vuoto perché è appena stato creato, o la ricezione di una grande quantità di dati dal log del Tmote dopo una interruzione temporanea dell'applicazione Java).
L'intervallo tra due controlli successivi e la durata del tempo di attesa iniziale, così come le dimensioni minime e massime della tabella su cui sono scritti i dati sono parametri modificabili nel file di configurazione.

[21] in realtà Watchdog causa la terminazione della LogStation, l'avvio successivo è causato dall'intervento di monit.

Configurazione

Ambiente di sviluppo

Lo sviluppo e il successivo utilizzo dell'applicazione Java richiedono che la variabile d'ambiente *CLASSPATH* sia correttamente impostata, in modo che includa le librerie MySQL e TinyOS; la procedura di installazione configura in maniera automatica i percorsi delle librerie. Se si volesse procedere "manualmente" alla configurazione dell'ambiente di sviluppo è necessario aggiungere a *CLASSPATH* i percorsi:

```
/usr/share/java/mysql-connector-java.jar
/opt/tinyos-2.1.1/support/sdk/java/tinyos.jar
```

È possibile che su sistemi diversi da quello utilizzato le librerie risiedano in posizioni differenti del file system: verificare i manuali di *TinyOS* e di *mysql-connector*.

File di configurazione

La copia del file di configurazione utilizzata dall'applicazione **deve** risiedere nella stessa directory che contiene i file .class compilati, cioè `~/InstallFiles/bin/` e **deve** chiamarsi `LogStation.config`.
Il file è organizzato in coppie *nome=valore*, una per ogni riga; le linee che cominciano con `#` sono considerate commenti.

Il primo gruppo di parametri controlla la Logstation e comprende

- le credenziali di accesso al database
- il path del *PID file*
- un flag per abilitare o disabilitare il logging degli errori
- il valore dell'intervallo con il quale il server invia l'orario al Tmote
- la dimensione dei buffer per le letture del termometro e del barometro[22]
- i fattori di correzione per la temperatura e la pressione[23]

Attenzione

Il *PID file* è scritto dalla LogStation ad ogni avvio e contiene il suo *process id* (PID).
Il path di questo file è definito anche nello script che avvia e termina il servizio di rilevazione meteorologica (`/etc/init.d/weather`) e nel file di configurazione di monit (`/etc/monit/conf.d/weather_java.conf`): per il corretto funzionamento del sistema, i tre file devono contenere lo stesso valore per questo parametro.

Attenzione

L'intervallo di `KEEPALIVE` con cui il server invia periodicamente il segnale orario è legato al parametro `SERVER_TIMEOUT` definito nel file di header `settings.h` dell'applicazione che gira

[22] vedi *BufferedFilter.java*

[23] vedi l'appendice sulle formule di conversione

sul Tmote: in particolare il `SERVER_TIMEOUT` **deve essere maggiore** del `KEEPALIVE` altrimenti, allo scadere del timeout, il Tmote non avrà ricevuto il segnale orario, quindi assumerà che il server si sia disconnesso ed entrerà nella modalità di logging fino alla ricezione del segnale orario successivo.
L'intervallo di `KEEPALIVE`, inoltre, non dovrebbe essere troppo ampio, pena una maggiore deriva del clock del Tmote; per tenere conto dei ritardi introdotti dal cavo, si è (arbitrariamente) deciso di porre il `SERVER_TIMEOUT` pari ad 1.5 volte il valore di `KEEPALIVE`, che è impostato a 30 secondi.

Il secondo gruppo di parametri riguardano la configurazione del Watchdog e controlla

- la dimensione minima che la tabella buffer può raggiungere
- la dimensione massima che tale tabella può raggiungere
- l'intervallo tra l'invio di due notifiche relative allo stesso errore
- l'intervallo di polling del Watchdog
- il tempo di sleep all'avvio del Watchdog

Attenzione
Questo secondo gruppo di intervalli è espresso in minuti.

Appendice: formule di conversione

Le formule che legano le grandezze fisiche lette dai sensori e le tensioni misurate ai loro capi sono riportate sui datasheet; le costanti che compaiono nelle formule provengono dalla documentazione o derivano da vincoli circuitali.

Convertire la lettura di un ADC in un valore di tensione

La formula generale per convertire il valore binario letto da un ADC in un valore di tensione è

$$V = V_{refneg} + \frac{\left(V_{refpos} - V_{refneg}\right) * ADCcount}{ADCFullScale}$$

dove V_{refpos} e V_{refneg} sono le tensioni di riferimento positiva e negativa a cui è collegato il convertitore, mentre *ADCFullScale* è il valore di fondo scala. Nel nostro caso V_{refneg} è sempre collegato a massa, quindi si può usare una formula più semplice

$$V = \frac{V_{ref} * ADCcount}{ADCFullScale}$$

Convertire un valore di tensione in un valore di illuminamento

I fotodiodi impongono il passaggio di una corrente direttamente proporzionale al valore di illuminamento in un una resistenza da 100 kΩ [vedi qui]. La tensione ai capi di questa resistenza è misurata da un ADC a 12 bit che usa una tensione di riferimento di 1.5 V.
Una volta noto il valore della tensione, è possibile risalire a quello della corrente applicando la prima legge di Ohm

$$V = RI$$

e dalla corrente all'illuminamento secondo la relazione [vedi qui]

$$Illuminamento: I = 100lux: Isc$$

Tutte le costanti che compaiono in queste formule sono state ricavate dai datasheet.

Leggere un valore di umidità relativa

La formula che permette di ottenere il valore dell'umidità relativa è

$$RH_{linear} = c_1 + c_2 * SO_{RH} + c_3 * SO_{RH}^2$$

dove SO_{RH} è il valore letto dall'ADC e c_1, c_2 e c_3 sono parametri indicati nel datasheet dell'igrometro che dipendono dal modello e dalla risoluzione del sensore.

È possibile compensare tale valore al variare della temperatura letta dal termometro integrato con la formula

$$RH = (Temperature - 25) * (t_1 + t_2 * SO_{RH}) + RH_{linear}$$

dove t_1 e t_2 sono due temperature di riferimento che dipendono sempre dalla risoluzione dell'igrometro, mentre *Temperature* si ottiene utilizzando la formula

$$Temperature = d_1 + d_2 * SO_T$$

sostituendo al posto di d_1 e d_2 due parametri che dipendono dalla scala (Celsius o Farenheit) utilizzata, dalla risoluzione e dalla tensione di alimentazione del sensore, mentre SO_T è il valore letto dall'ADC associato al termometro.

Convertire un valore di tensione in un valore di temperatura

Il termometro di precisione impone il passaggio di una corrente direttamente proporzionale alla temperatura attraverso una resistenza da 5487 Ω (il valore della resistenza montata sulla basetta è stato misurato con un multimetro); in particolare, il fattore di proporzionalità è di 1μA per ogni K.
Per convertire la tensione in temperatura, quindi si calcolerà la corrente applicando la prima legge di Ohm

$$V = RI$$

e dalla corrente, applicando

$$T = -273.15 + \frac{I}{1\mu A/K}$$

sarà possibile ottenere la temperatura espressa in gradi centigradi.

Compensazione della temperatura
La temperatura influenza anche il valore della resistenza collegata al sensore; per compensare questo comportamento, il valore in Kelvin ottenuto dalla conversione è moltiplicato per un coefficiente α prima di essere ulteriormente convertito in gradi centigradi.
Il valore del coefficiente è stato determinato confrontando il comportamento della nostra stazione meteorologica con quello di una stazione meteorologica considerata "affidabile".

$$T = -273.15 + \frac{I}{1\mu A/K} * \alpha$$

Convertire un valore di tensione in un valore di pressione
Il barometro è collegato in parallelo ad un partitore resistivo formato da due resistenze identiche: la tensione letta dall'ADC sarà dunque la metà di quella presente ai capi del sensore, quindi sarà necessario moltiplicarla per due prima di applicare la formula

$$P = \frac{\frac{V}{5} + 0.095}{0.009}$$

che restituisce il valore della pressione atmosferica misurato in kPa. Per ottenere il corrispondente valore in hPa o in mbar è sufficiente moltiplicare il risultato così ottenuto per 10.

Compensazione della pressione atmosferica
Anche in questo caso il valore letto dai sensori è compensato con un coefficiente moltiplicativo ricavato confrontando la nostra stazione meteorologica una di riferimento.

$$P = \frac{\frac{V}{5} + 0.095}{0.009} * \beta$$

Convertire un valore binario in millimetri di pioggia
Ogni 0.2794 mm di pioggia caduti, il pluviometro incrementa di una unità il contatore della stazione meteorologica.

Convertire un valore binario nella direzione del vento
Il programma in esecuzione sul Tmote definisce una tabella di conversione tra la tensione misurata ai capi della banderuola e la direzione del vento[24] ed implementa un algoritmo di look up per restituire il valore corretto all'applicazione Java.
Gli angoli sono misurati in senso orario a partire da Nord (cui corrisponde il valore uno) con una risoluzione di 22.5°.
Per ottenere la direzione in gradi è sufficiente applicare la formula

[24] si tratta della struttura `compass_rose` definita nel file `settings.h`

$$D = 22.5° * (direction - 1)$$

Offset sulla direzione del vento
È possibile specificare un offset in gradi da aggiungere alla direzione del vento per correggere le misure dovute ad una orientazione della banderuola che non è perfettamente allineata con i punti cardinali.

Misurare la velocità del vento
Secondo il datasheet dell'anemometro, una corrente d'aria che si muove alla velocità di 2.4 Km/h produce un impulso al secondo ai capi del sensore.

$$V = V_0 * \frac{count}{T}$$

Contando il numero degli impulsi, dividendoli per la durata dell'intervallo di tempo considerato e moltiplicandoli per 2.4 Km/h è possibile conoscere la velocità media del vento.

Database

Configurazione

I file di configurazione del database MySQL forniti al momento dell'installazione non hanno subito modifiche; lo script di installazione provvede ad aggiungere nella directory `/etc/mysql/conf.d/` il file `pikachu.cnf` che contiene le direttive per l'avvio dello scheduler allo startup del DBMS.

Utenti

root
è responsabile della creazione del database e delle altre utenze durante la fase di installazione, come specificato nel file `~/InstallFiles/source/mysql.sql`

weatherJava
è l'utente associato all'applicazione Java: si occupa della creazione di tutti gli altri oggetti del database (tabelle e procedure), del popolamento della base di dati e dello scheduling delle operazioni di compressione e pulizia periodica delle tabelle; gode dei privilegi di `SELECT, INSERT, UPDATE, DELETE, CREATE, EXECUTE, CREATE ROUTINE, EVENT` sul database weather; è inoltre in grado di impostare variabili globali, avviare lo scheduler di mysql e controllare i processi in esecuzione nel DBMS mediante i privilegi SUPER, PROCESS su tutti i database.
Questa utenza necessita di così tanti privilegi in quanto il server deve creare la struttura della base dati in fase di installazione. Oltre a questo il server crea le procedure invocate dai due eventi i quali a loro volta vengono eseguiti con le credenziali dell'utente weatherJava. Il privilegio di DELETE è infatti richiesto proprio dalla procedura che aggrega i dati e li sposta nella tabella record e history.

weatherApache
è l'utente associato all'applicazione web e gode dei soli privilegi di lettura sulle tabelle `buffer`, `record` e `history`.

Struttura del database

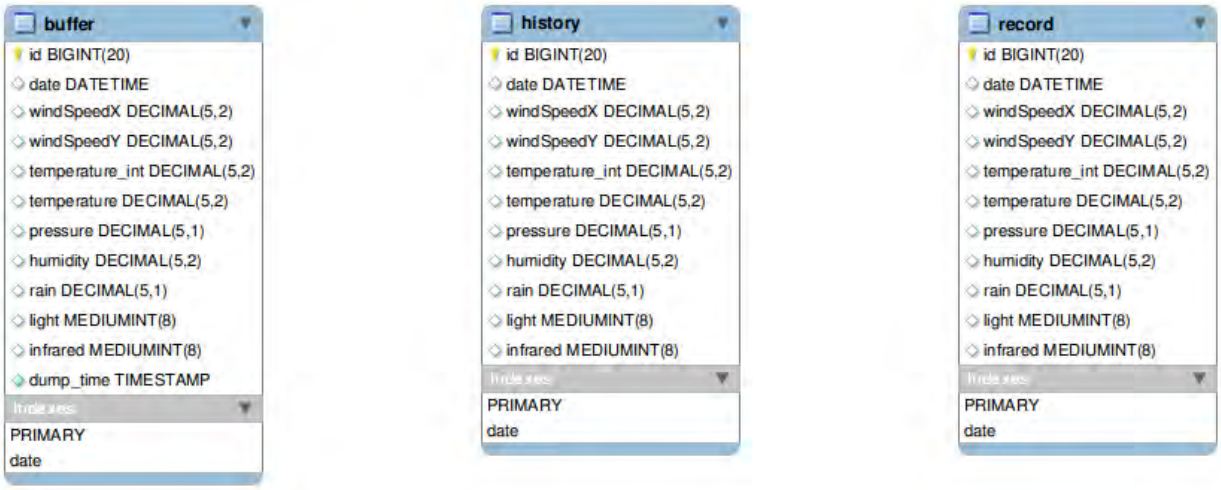
La base di dati è composta dalle tabelle:

`buffer`
contiene le ultime rilevazioni della stazione meteorologica. I dati grezzi sono mantenuti in questa tabella al più per dieci minuti dal caricamento prima di essere cancellati.

`record`
contiene i dati dell'ultimo anno aggregati in intervalli di cinque minuti: si tratta della media aritmetica delle misure effettuate, ad eccezione dei valori relativi alle precipitazioni (colonna `rain`) che sono sommate.
`history`
contiene i dati degli anni precedenti, aggregati per giorno.

La tabella buffer ha una colonna in più rispetto alle altre due: `dump_time` è il timestamp dell'operazione di caricamento dei dati sul server e serve alla procedura che si occupa dell'aggregazione dei dati.

Schema E-R



Eventi

La procedura *`five`* si occupa della compressione dei dati grezzi e della pulizia periodica della tabella `buffer`: ogni cinque minuti la routine calcola la media aritmetica delle ultime rilevazioni **caricate** sul server (i dati relativi alle precipitazioni costituiscono un'eccezione, in quanto sono sommati), scrive i valori aggregati così ottenuti nella tabella `record` e cancella tutte le tuple che riportano un `dump_time` più vecchio di cinque minuti rispetto al timestamp corrente (e.g. alle 10:30 `five` compatterà i dati caricati tra le 10:25 e le 10:30 e cancellerà tutte le rilevazioni che risalgono a prima delle 10:25).
La compressione di un set di dati e la sua eliminazione avvengono, quindi, in due esecuzioni successive di `five`: in questo modo evitiamo di cancellare l'*ultima lettura*, in modo che sia disponibile per l'applicazione web.

Attenzione!
La procedura *`five`* seleziona i dati che saranno compattati in base al timestamp con il quale sono stati caricati sul server (`dump_time`), ma li aggrega secondo il timestamp di generazione (`date`).

Se la `LogStation` dovesse terminare in modo inaspettato, infatti, i dati registrati dal Tmote sulla memoria flash potrebbero essere scritti nel database più di dieci minuti dopo la loro generazione: se `five` confrontasse unicamente i timestamp di creazione, questi dati non sarebbero mai aggregati, anzi, verrebbero cancellati immediatamente, vanificando del tutto le strategie di logging implementate sul Tmote!

La procedura *`historic`* si occupa della seconda compressione dei dati e della pulizia periodica della tabella `record`.
Ogni notte a mezzanotte la routine calcola la media aritmetica delle rilevazioni dell'anno precedente (ad eccezione dei dati relativi alle precipitazioni, che invece vengono sommati), scrive i valori aggregati così ottenuti nella tabella `history` e cancella dalla tabella `record` tutte le tuple che portano una data più vecchia di un anno di quella attuale.

Note

Il corpo delle procedure `five` e di `historic` è visualizzabile eseguendo

```
SELECT event_name, event_definition
  FROM information_schema.events;
```

dal client di MySQL; in realtà le procedure richiamano altre due routine, rispettivamente *`buffer_to_record`* e *`record_to_history`*, il cui codice è visibile eseguendo

```
SELECT routine_name, routine_definition
  FROM information_schema.routines;
```

Ogni esecuzione di queste procedure avvia una nuova transazione, risolvendo così alcuni comportamenti anomali riscontrati in fase di sviluppo (lo scheduler di MySQL avviava più istanze di `five`, ognuna delle quali produceva una copia - assolutamente ridondante - dei risultati nella tabella `record`) e durante il testing (procedure che restavano bloccate per un tempo indefinito).
Le procedure **non funzionano** se lo scheduler di MySQL non è in esecuzione; il file `pikachu.cnf` in `/etc/mysql/conf.d/` contiene le direttive per l'avvio automatico dello scheduler allo startup del DBMS.

Ulteriori informazioni sugli eventi e sullo scheduler sono disponibili sul manuale di MySQL.

Applicazione Web

Configurazione

I parametri (hostname, database, username e password) con cui l'applicazione web si collega alla base di dati sono definiti all'interno del file `www/misc/include.php`.
La configurazione del web server è quella di default, fornita al momento dell'installazione del software, ad eccezione del file `/etc/apache2/sites-available/default` che è stato modificato per ragioni di sicurezza, in modo da disabilitare il listing delle directory.

Pagine principali

Index.php
La home page del sito è organizzata in tre sezioni: la prima mostra l'ultima rilevazione della stazione meteorologica, completa di data e ora; la seconda e la terza permettono di visualizzare, sia in forma tabellare che grafica, le rilevazioni medie rispettivamente degli ultimi 60 minuti e delle ultime 24 ore.
Le informazioni mostrate in questa pagina si aggiorneranno automaticamente ogni 30 secondi.

Stats.php
Questa pagina permette al visitatore di recuperare dal database tutte le letture della stazione meteorologica che si riferiscono ad un particolare intervallo di tempo: dopo aver specificato la data di inizio e quella di fine osservazione, l'utente potrà vedere i dati richiesti sia in forma tabellare che in forma grafica (in modo simile a quanto accade nella home).
Le rilevazioni relative a periodi di durata pari o inferiore a 7 giorni saranno presentate aggregate per ora, quelle relative a periodi più lunghi di una settimana saranno aggregate su base giornaliera.

Panel.php
Permette di leggere l'ultima rilevazione della stazione meteorologica simulando un display digitale; la pagina, che è anche utilizzata per fornire la *modalità chiosco*[25], si aggiornerà automaticamente ogni 5 secondi.

Backend

misc/include.php
Oltre a parametri di connessione al database, contiene alcune funzioni per la generazione degli elementi comuni alle varie pagine del sito, quali l'intestazione, la barra di navigazione o il piè di pagina.

misc/ajax.php
Si tratta del back-end AJAX dell'applicazione web: esegue le interrogazioni sul database e formatta i risultati per le altre pagine del sito.

[25] vedere il capitolo Installazione alla voce Modalità chiosco

JQuery, jQueryUI e Flot
Le librerie Javascript utilizzate dall'applicazione web risiedono in `/var/www/javascript/` .

Validazione
Il sito ha superato la validazione del W3C per lo standard HTML5.

Grafici

I grafici delle rilevazioni sono disegnati dallo script `www/javascript/mygraph.js` che fa uso delle funzionalità di flot, un plugin per jQuery.
Al suo interno definisce due funzioni:

- `draw(source, checklist)` è la funzione principale, richiamata dalle pagine web per tracciare i grafici delle rilevazioni.
  Il parametro `source` indica l'elemento che contiene sia la tabella da cui saranno estratti i dati, sia il pannello che ospiterà i grafici; `checklist` è la lista delle grandezze fisiche che l'utente ha scelto di visualizzare, spuntando o meno un apposito array di checkbox. La funzione definisce un'etichetta, un colore ed un sistema di assi cartesiani per ogni grandezza fisica, recupera i dati dalla colonna corrispondente della tabella delle osservazioni e richiama il metodo `plot` del plugin che si occupa di disegnare i grafici e renderne interattivi i punti.
- `showTooltip(x, y, contents)` viene richiamata ogni volta che l'utente clicca uno dei punti del grafico; mostra un pannello contenente il nome della grandezza fisica, l'ora a cui la misura si riferisce e il valore rilevato dal sensore.

Attenzione
I grafici mostrano solo la velocità del vento, senza indicarne la direzione.

Strumenti di monitoring

Nella sezione relativa all'applicazione LogStation si parla di componenti di *monitoring*. Sono strutture di controllo interne che verificano il funzionamento del servizio stesso. Oltre a queste è stato installato un pacchetto software esterno chiamato Monit. Esso verifica costantemente il corretto funzionamento di determinati servizi e eventualmente interviene in caso di stato di esecuzione anomala. I parametri osservati sono prevalentemente il tempo CPU, la memoria allcoata, la presenza o meno di determinati file ecc...
Monit fornisce inoltre un'interfaccia web tramite la quale è possibile visualizzare lo stato delle applicazioni monitorate e eventualmente riavviarle. Attualmente il sito è accessibile tramite https://meteo.dsi.unimi.it:8080

Installazione

Per installare Monit è sufficiente usare il gestore di pacchetti del sistema con il comando:

```
sudo apt-get install monit
```

L'arresto e l'avvio avvengono come per altri servizi:

```
sudo /etc/init.d/monit stop
```

Nota Bene
Qualora si volesse arrestare definitivamente il servizio weather è necessario arrestare prima Monit, altrimenti esso noterebbe che weather non è in esecuzione e lo riavvierebbe.

Configurazione

Per la configurazione di monit consultare il manuale al sito ufficiale riportato nella sezione Bibliografia di questo documento. Il file di configurazione principale è `/etc/monit/monitrc` mentre in `/etc/monit/conf.d` sono presenti i file per la configurazione dei singoli servizi (Apache2, MySQL e Weather)

File di log

Oltre a intraprendere azioni specifiche in casi di malfunzionamento Monit tiene traccia del suo stato e degli errori avvenuti nel file di log `/var/log/monit.log`.

Bibliografia

Tmote
http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

Sensiron SHT11 (Igrometro)
http://www.sensirion.com/en/01_humidity_sensors/02_humidity_sensor_sht11.htm

Fotodiodi
http://jp.hamamatsu.com/resources/products/ssd/pdf/s1087_etc_kspd1039e01.pdf

Termometro
http://www.analog.com/static/imported-files/data_sheets/AD592.pdf

Barometro
http://cache.freescale.com/files/sensors/doc/data_sheet/MPXA6115A.pdf?fpsp=1

Banderuola, anemometro e pluviometro
https://www.argentdata.com/files/80422_datasheet.pdf

TinyOs
http://www.tinyos.net
http://docs.tinyos.net - documentazione tecnica
http://docs.tinyos.net/tinywiki/index.php/TEPs - TinyOS Enhancement Proposals
http://mail.millennium.berkeley.edu/pipermail/tinyos-help/ - TinyOS help mailing list
http://www.tinyos.net/tinyos-2.1.0/doc/nesdoc/telosb/ - reference guide specifica per hardware Tmote

Eclipse
http://www.eclipse.org/

Yeti2 - TinyOs IDE basata su Eclipse
http://tos-ide.ethz.ch/wiki/index.php

Linguaggio di programmazione NesC
http://it.wikipedia.org/wiki/NesC
http://nescc.sourceforge.net/papers/nesc-ref.pdf
http://nescc.sourceforge.net/

Java
http://www.oracle.com/technetwork/java/javase/documentation/index.html
http://docs.oracle.com/javase/6/docs/api/ - reference Java 6

MySQL
http://dev.mysql.com/doc/refman/5.5/en/index.html

Ubuntu server
https://help.ubuntu.com/11.04/serverguide/C/index.html

Bash scripting

http://tldp.org/LDP/abs/html/

Monit
http://mmonit.com/monit/documentation/

jQuery
http://jquery.com/
http://code.google.com/p/flot/ - Flot: attractive Javascript plotting for jQuery
http://people.iola.dk/olau/flot/examples/ - Flot examples