UNIVERSITÀ
DEGLI STUDI
DI PADOVA

A.A. 2015/2016

RELAZIONE DI LABORATORIO INGEGNERIA INFORMATICA

Pasqualini Alessandro
alessandro.pasqualini.1105@gmail.com

INTRODUZIONE CON OBIETTIVI E MOTIVAZIONI DEL PROGETTO

Il progetto Lewe2.0 nasce dalle ceneri di un precedente progetto denominato Lewe (https://github.com/alessandro1105/Lewe) con l'obiettivo di evolverne codice e hardware, mantenendone, invece, inalterati gli scopi e gli obiettivi.

Lewe è stato concepito per raccogliere e rielaborare varie tecnologie di telecomunicazione nel mondo dell'IOT (Internet Of Things) dove vari dispositivi devono comunicare in tempo reale per la realizzazione di un obiettivo comune, che spazia dalla lettura di sensori al telecontrollo di elettrodomestici.

Attualmente nell'ambito dell'IOT non esiste un protocollo di comunicazione comunemente accettato che sia al contempo abbastanza leggero, flessibile, efficiente e sicuro da integrarsi agli svariati contesti dove l'IOT trova la sua applicazione.

Lewe (e Lewe2.0) nasce dunque con l'idea di realizzare un protocollo di comunicazione che rispecchi quanto più possibile le caratteristiche sopra citate partendo da un contesto applicativo semplificato, ma non banale, come quello della rilevazione di alcuni dati biometrici attraverso un bracciale dotato di sensori e l'invio degli stessi ad una applicazione per smartphone Android.

Il contesto non risulta in alcun modo banale per il fatto che gran parte delle applicazioni quotidiane dell'IOT prevedono proprio il monitoraggio da remoto di sensori sulle cui letture vengono intraprese delle azioni di telecontrollo di qualsiasi genere e tipologia.

Ricapitolando, il progetto Lewe2.0 prevede la realizzazione di un bracciale (prototipo di fitness tracker) dotato di alcuni sensori le cui letture devono essere inviate, attraverso il protocollo di comunicazione denominato Jack, ad una applicazione Android.

Il protocollo stesso è in realtà un prototipo e lo scopo principale di Lewe2.0 è quello di comprendere i requisiti che lo stesso dovrà avere, partendo da una struttura simile al JSON (JavaScript Object Notation) che risulta molto più semplice da comprendere per un essere umano rispetto ad un protocollo basato su codice binario tipo CoAP (Constrained Application Protocol).

ARCHITETTURA E PROGETTAZIONE DELL'APPLICAZIONE

Il progetto si suddivide in tre blocchi principali: il bracciale, l'applicazione per smartphone Android e il protocollo di comunicazione Jack.

a) b)

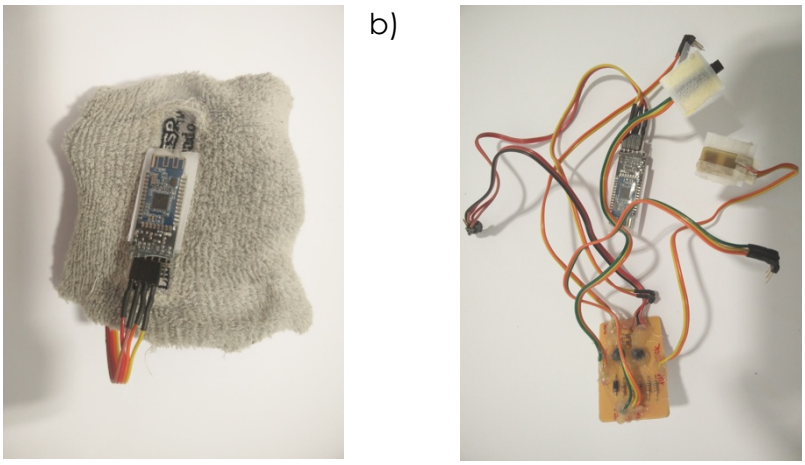


Fig. 0a: Foto bracciale

Fig. 0b: Foto hardware del bracciale

Il bracciale è composto da Fishino UNO (un clone di Arduino che integra RTC e WiFi) e da una basetta autocostruita contenente tutto l'hardware per pilotare i sensori di temperatura e GSR (Galvanic Skin Response).

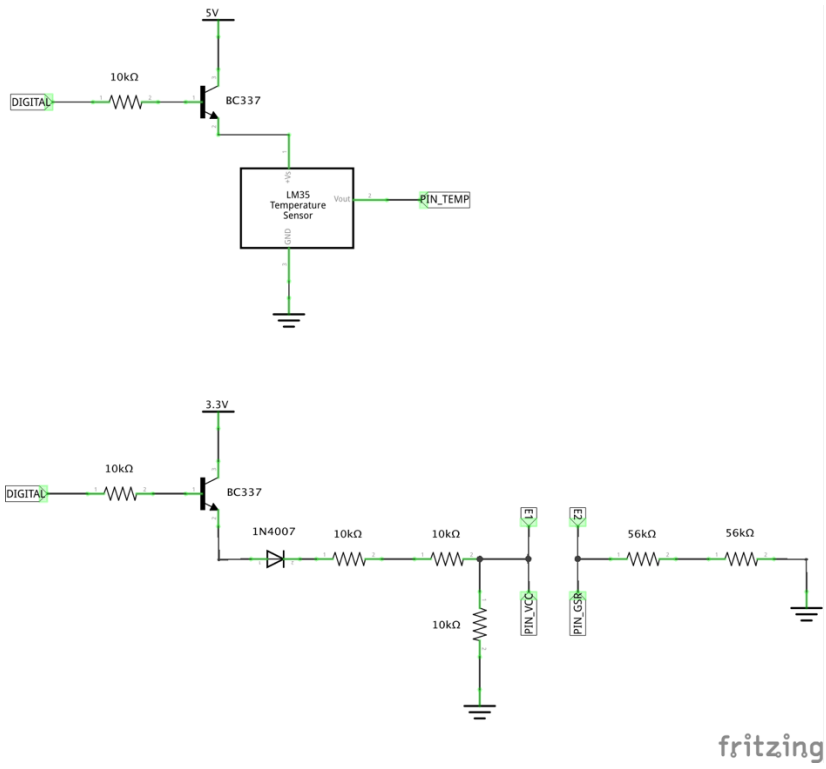


Fig. 1: Schema completo del circuito usato per pilotare i sensori.

Il sensore di temperatura è un LM35DZ e fornisce come output un valore di tensione proporzionale alla temperatura (10mV/°C).

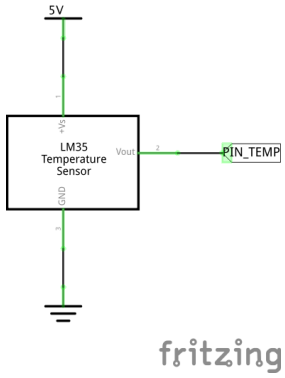


Fig. 2: Schema di connessione del sensore di temperatura LM35DZ.

La temperatura è ricavata attraverso la formula:

$$Temperatura = \frac{analogRead(PIN_{Temp}) * V_{REF}}{1023} * 100$$

dove V_{REF} è la tensione di riferimento per l'ADC (Analog/Digital Converter) di Fishino UNO, che per questa applicazione è stato impostato a 1.1V.

Il sensore GSR (Galvanic Skin Response) è formato da due elettrodi a contatto diretto con la pelle, ai quali viene applicata una piccolissima tensione, impercettibile dal soggetto indossatore del bracciale, attraverso la quale è possibile misurare la resistenza della pelle che varia in relazione a stimoli emotivi che agiscono sulle ghiandole sudoripare.

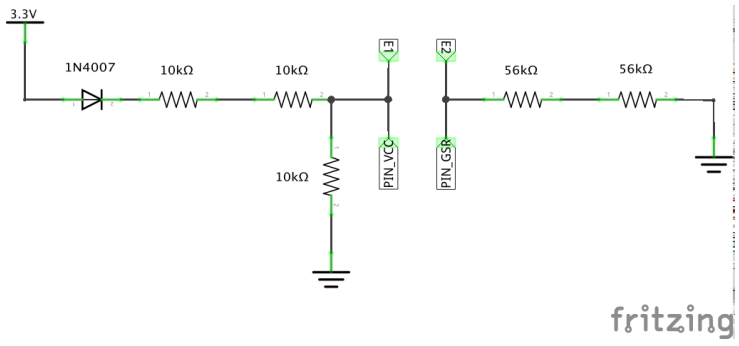


Fig. 3: Schema del sensore GSR.

La lettura del sensore in Ohm non è di facile comprensione e per questo motivo si è preferito convertire questo valore in percentuale per rendere più agevole la sua lettura da parte di chiunque. La formula usata per la conversione è la seguente:

$$GSR = \frac{100 * analogRead(PIN_{GSR})}{analogRead(PIN_{Vcc})}$$

Il rapporto $\frac{analogRead(PIN_{GSR})}{analogRead(PIN_{Vcc})}$ è usato, anche, per autocalibrare il sensore, rendendolo immune ad eventuali cali di tensione dovuti all'esaurimento progressivo dell'alimentazione.

Poiché il bracciale è alimentato interamente a batteria è stato anche previsto un meccanismo per il risparmio energetico che consente di attivare i sensori solo per la lettura e di disattivarli al termine della stessa riducendone quindi il consumo. Il circuito si avvale di un transistor BC337 in modalità ON/OFF, ovvero è usato come interruttore pilotato da un segnale digitale proveniente dal MCU (MicroController Unit).

a) b)

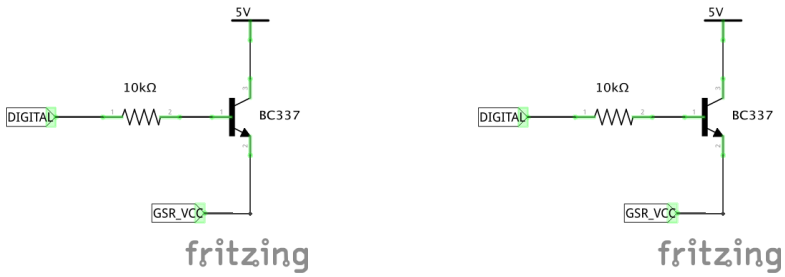


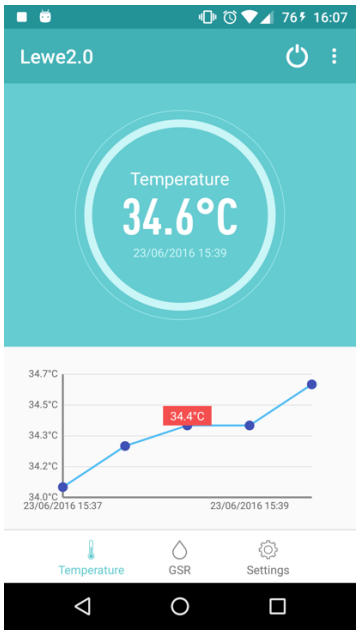
Fig. 4a: Schema di risparmio energetico per il sensore di temperatura LM35DZ.

Fig. 4b: Schema di risparmio energetico per il sensore GSR.

Il modulo usato per la connessione bluetooth con l'applicazione per smartphone Android è un HM-10. E' un modulo BLE (Bluetooth Low Energy) che simula una connessione seriale attraverso un servizio e una caratteristica condivisa tra RX e TX. Esso è dotato di un'interfaccia UART che consente, oltre che l'invio di dati attraverso la connessione seriale simulata, l'esecuzione di comandi (AT), alcuni dei quali sono stati usati per la configurazione iniziale (si rimanda al datasheet del modulo per una completa trattazione dei comandi e delle possibili configurazioni).

L'applicazione per smartphone Android permette la visualizzazione delle letture dei sensori posti nel bracciale.

a)



b)

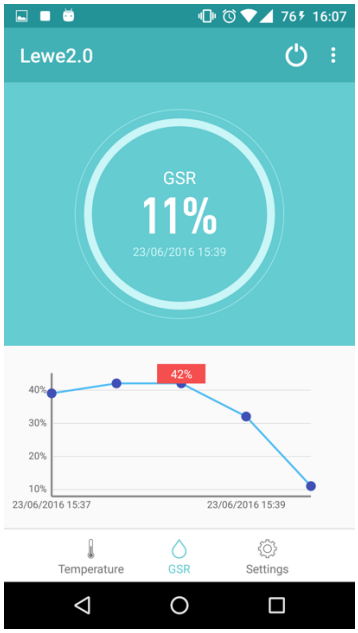
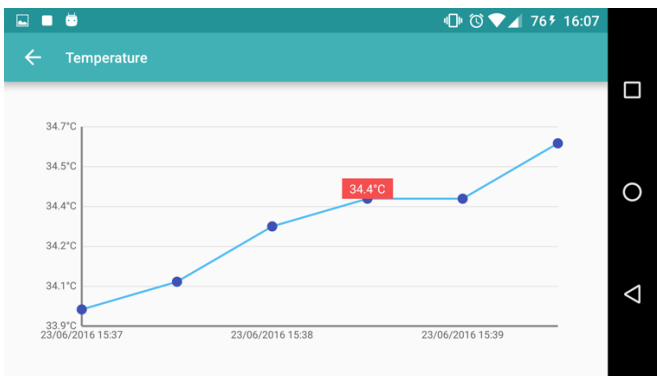


Fig. 8a: Screenshot main activity sezione temperatura.

Fig. 8b: Screenshot main activity sezione GSR.

La main activity si configura in due sezioni: la prima visualizza l'ultima lettura ricevuta dei sensori (insieme alla data e ora di rilevazione), mentre la seconda mostra graficamente le ultime 5 letture. Cliccando sul grafico è possibile accedere ad un'altra activity che permette di visionare, sempre attraverso un grafico, tutte le letture ricevute dal bracciale.

a)



b)

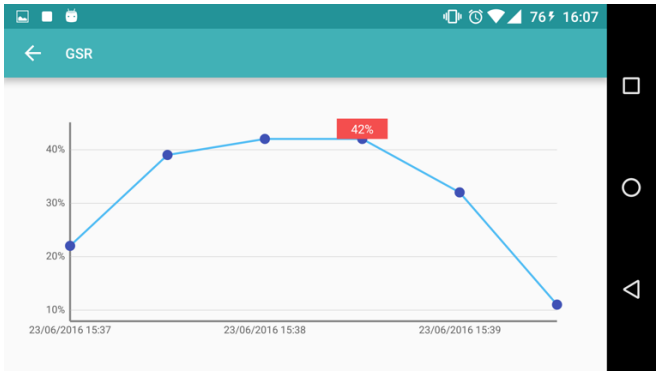
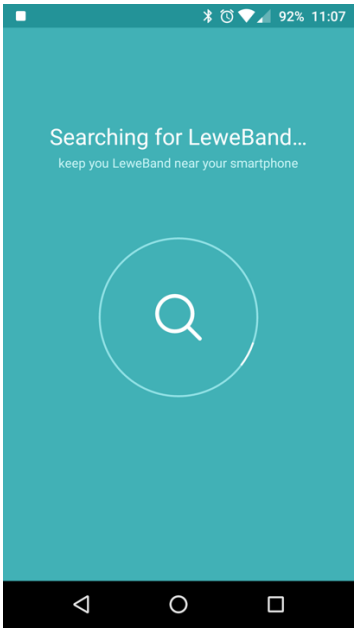


Fig. 8a: Screenshot schermata grafico temperatura.

Fig. 8b: Screenshot schermata grafico GSR.

L'associazione tra applicazione e bracciale avviene attraverso una schermata specifica, dove l'applicazione si connette al primo device bluetooth chiamato "LW2.0" e ne ricorda l'indirizzo MAC per potersi connettere al successivo utilizzo.

a)



b)

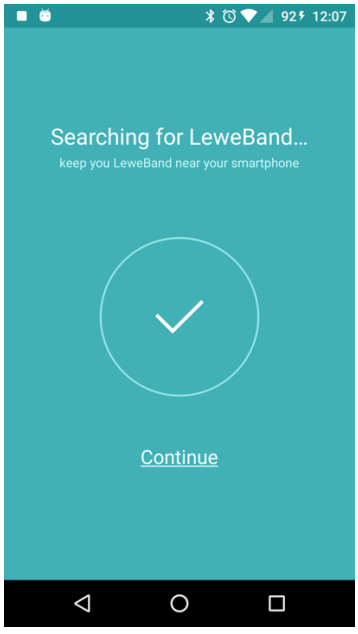


Fig. 8a: Screenshot schermata di ricerca di LeweBand.

Fig. 8b: Screenshot di associazione completata.

IL PROTOCOLLO DI COMUNICAZIONE JACK

Il protocollo di comunicazione Jack (JSON ACK) è stato basato sul formato JSON data la semplicità e la facilità di lettura tipiche di questo formato, al quale sono state però applicate delle regole per la composizione dei messaggi.

Il protocollo garantisce una connessione affidabile tra mittente e destinatario implementando un sistema di conferme e mettendo a disposizione due tipologie di messaggi: *DATA* e *ACK*.

IL MESSAGGIO DATA

Questa è la tipologia di messaggio principale, ovvero quella che contiene i dati che l'host mittente desidera inviare all'host destinatario.

Data l'importanza del contenuto di questo messaggio, esso necessita di una conferma dal duplice significato: il messaggio è stato ricevuto ed è nel formato corretto, ovvero della tipologia DATA.

La struttura del messaggio è riportata di seguito:

```
{
	"id": ID,
	"type": "data",
	"val":
		{
			"key1": val1,
			"key2": "val2,
		}
}
```

dove ID indica l'identificativo univoco del messaggio.

L'oggetto "val" contiene tutti i dati che il messaggio deve trasportare nella forma "chiave: valore", esso non è obbligatorio e può anche essere omesso.

IL MESSAGGIO ACK

Il messaggio ACK è il messaggio di conferma che il destinatario invia al mittente per confermare la ricezione e validità di un messaggio DATA ricevuto.

Il destinatario in mancanza della conferma, allo scadere di un determinato timer, rinvierà il messaggio contenente i dati fino alla conferma dello stesso.

La struttura del messaggio ACK è riportata di seguito:

```
{
	"id": ID,
	"type": "ack"
}
```

dove ID è l'identificativo del messaggio da confermare.

SVILUPPO E IMPLEMENTAZIONE

LIBRERIE ARDUINO IDE

Il firmware del bracciale è stato sviluppato in C/C++ insieme alla realizzazione della libreria Hashmap (basata sui sorgenti di Alexander Brevig per il progetto Wiring), in quanto, l'IDE di Arduino ne risulta sprovvisto ed è di essenziale importanza per la libreria Jack che implementa il protocollo omonimo.

La libreria Jack è composta da 3 classi: Jack, JData e JTransmissionMethod.

JTRANSMISSIONMETHOD

JTransmissionMethod consiste di una classe astratta che contiene i metodi necessari per gestire un mezzo di trasmissione, ovvero il canale fisico che il protocollo di comunicazione dovrà usare per l'invio e la ricezione dei messaggi. Naturalmente essendo una classe astratta i metodi sono privi di corpo e consistono sono dell'intestazione; al momento dell'implementazione della classe di gestione di uno specifico canale comunicativo sarà dunque necessario estendere questa classe e sviluppare il corpo dei seguenti metodi:

```
virtual size_t receive(char *buffer, size_t size)

virtual void send(char *message, size_t length)

virtual size_t available()
```

Il funzionamento è molto semplice: *send* ha il compito di inviare, attraverso il mezzo di comunicazione, il messaggio passatogli come argomento, *available* deve restituire il numero di caratteri che ha ricevuto fino al momento della chiamata, mentre *receive* ha il compito di inserire nel buffer (passato come argomento) il primo messaggio disponibile e di restituire la sua lunghezza.

JDATA

La classe JData è un contenitore di dati creato appositamente in modo da ridurre i requisiti di memoria di questa implementazione del protocollo. Essa si basa sulla libreria ArduinoJson (https://github.com/bblanchon/ArduinoJson) che consiste di un'implementazione leggera del formato JSON appositamente costruita per la famiglia di schede di sviluppo Arduino.

JData dispone di metodi set (uno per ogni tipo primitivo) e get (unico per tutti) usati per l'aggiunta e il prelievo dei dati inseriti nel contenitore. Il metodo get sfrutta il tipo JsonVariant (messo a disposizione dalla libreria ArduinoJson) che consente un cast implicito in base al tipo del target (per maggiori informazioni sul funzionamento si invita a leggere il wiki della libreria all'indirizzo https://github.com/bblanchon/ArduinoJson/wiki)

La classe è stata sviluppata in modo che l'aggiunta di un dato avvenga direttamente in un oggetto JsonObject (oggetto della libraria ArduinoJson) da cui successivamente verrà costruito il messaggio Jack.

JACK

La classe Jack è la classe principale della libreria e ha il compito di gestire il protocollo omonimo.

Il costruttore della classe è il seguente:

```
Jack(JTransmissionMethod &mmJTM, void (*onReceive)(JData &, long), void
(*onReceiveAck)(long), long (*getMessageID)())
```

esso richiede un'istanza del mezzo di comunicazione (mmJTM) e tre puntatori a funzioni, rispettivamente un handler per la gestione dell'evento di ricezione di un messaggio, un handler per la gestione della ricezione di messaggi ACK e un puntatore a funzione che deve restituire un codice unico ad ogni invocazione (questo codice è utilizzato come ID per i messaggi inviati).

La classe prevede un altro costruttore che permette di specificare un valore per il timer di invio dei messaggi (il tempo di attesa tra due invii) e un timer per il polling del mezzo di comunicazione (il tempo di attesa tra due interrogazioni).

Una volta istanziato un oggetto della classe Jack è necessario avviare il polling chiamando il metodo *start* (è possibile anche interromperlo utilizzando *stop*).

In Arduino (e quindi Fishino UNO) non è possibile l'esecuzione di thread e processi per la mancanza di un sistema operativo, dunque il polling e l'invio dei messaggi allo scadere dei rispettivi timer è affidato ad un membro funzionale della classe chiamato *loop* che necessità di essere chiamato ripetutamente all'interno del firmware (la chiamata è stata posta dentro la funzione *loop* predisposta dall'ambiente di sviluppo Arduino IDE).

Ogni volta che il metodo *loop* viene invocato viene controllata la presenza di messaggi in attesa nel canale di comunicazione provenienti da un altro host e, se presenti, vengono prelevati e gestiti, oltre a ciò vengono inviati tutti i messaggi ancora in attesa di conferma.

L'invio dei messaggi da parte dell'utente utilizzatore della libreria è affidato al seguente metodo:

```
long send(JData &message)
```

Il metodo preleva l'istanza della classe JsonObject, contenuta dentro il parametro *message* passato durante l'invocazione del metodo, e aggiunge l'identificativo del messaggio usando come valore il risultato della funzione a cui fa riferimento getMessageID (il puntatore a funzione è stato precedentemente passato al costruttore durante la creazione dell'oggetto della classe Jack).

Una volta aggiunte le proprietà aggiuntive all'oggetto JsonObject esso viene codificato in JSON e posto nel buffer contenente i messaggi in attesa di invio; esso verrà inviato allo scadere del timer di invio dei messaggi.

Nel costruttore della classe Jack sono richiesti anche due funzioni handler che hanno il compito di gestire i due eventi principali: la ricezione di un messaggio e la ricezione di una conferma per un messaggio precedentemente inviato; esse vengono invocate dopo che l'elaborazione di un messaggio prelevato dal mezzo di trasmissione ha rivelato la tipologia del messaggio stesso.

Se la tipologia è un messaggio DATA viene istanziato un oggetto della classe contenitore JData e passato all'handler *onReceive* insieme all'identificativo del messaggio; se invece il messaggio è una conferma viene prelevato l'identificativo e viene passato all'handler *onReceiveAck*.

FIRMWARE DEL BRACCIALE

Il firmware del bracciale ha il compito di gestire l'hardware dei sensori e di generare i messaggi da inviare all'applicazione per smartphone Android.

Il codice principale è contenuto all'interno di due funzioni necessarie per lo sviluppo attraverso la tecnologia Arduino, ovvero *setup* e *loop*.

La funzione *setup* ha il compito di configurare l'hardware e di avviare il polling del mezzo di comunicazione; essa invoca alcune funzioni disegnate specificatamente per l'hardware di cui è dotato il bracciale: *setupSensor* e *setupBluetooth*.

La prima abilità i pin di comando dei sensori in modalità di output (il segnale inviato tramite questi pin dovrà comandare l'accensione e lo spegnimento dei sensori) e inizializza il modulo RTC, impostando l'ora di compilazione nel caso che lo stesso non fosse attivo.

La seconda funzione di configurazione inizializza la seriale software necessaria per comunicare con il modulo bluetooth HM-10.

Per pilotare i sensori sono state implementate alcune funzioni: sleepSensor e wakeupSensor, che come rivela in nome, hanno il compito rispettivamente di spegnere e di accendere i sensori.

Le letture dei sensori vengono prelevate grazie all'ausilio delle funzioni: getGSR, getTemperature e getTimestamp, il cui compito è quello di prelevare il dato dai sensori, di elaboralo (eliminando eventualmente il rumore intrinseco) e di restituirlo.

La seconda funzione principale, *loop*, dopo aver eseguito il metodo omonimo dell'oggetto della classe Jack, abilita i sensori, invia il messaggio allo smartphone Android contenente le letture dei sensori e disabilita i sensori.

APPLICAZIONE PER SMARTPHONE ANDROID

L'applicazione per smartphone Android è stata costruita utilizzando l'ambiente di sviluppo Androd Studio ed è stata sviluppata in Java, con l'ausilio delle API messe a disposizione dal sistema operativo e di una libreria per la realizzazione dei grafici in essa contenuti (https://github.com/bblanchon/ArduinoJson).

Lo sviluppo della stessa esula dai contesti di questo corso e dunque non verrà illustrata la sua implementazione (è comunque disponibile la documentazione prodotta).

VALUTAZIONE E COLLAUDO

La natura del progetto si è prestata ad uno sviluppo modulare in cui ogni componente è stato testato autonomamente prima di essere integrato nel sistema completo, che a sua volta è stato collaudato testando ogni sua funzionalità; particolare attenzione è stata posta all'elaborazione dei messaggi scambiati attraverso il protocollo di comunicazione.

L'hardware di gestione dei sensori è stato implementato utilizzando, come supporto di prototipazione, una basetta sperimentale (breadboard) e successivamente, ottenuto il circuito definitivo, adeguatamente testato, è stato costruito il PCB finale utilizzando il software Fritzing per generare lo schema del circuito.

Il firmware del bracciale è stato testato grazie all'ausilio di stringhe di log inserite all'interno dello stesso e stampate utilizzando la seriale hardware presente in Fishino UNO e visionate attraverso lo strumento *Monitor seriale* fornito all'interno dell'IDE di Arduino.

I log sono stati stampati utilizzando il seguente snippet di codice:

```
#ifdef DEBUG
  Serial.print(F("log"));
#endif
```

Per disabilitare tutto il codice di collaudo, per la versione definitiva del firmware, è stata semplicemente eliminata (commentata) la direttiva al preprocessore chiamata DEBUG.

Tutte le librerie sviluppate per fornire le funzionalità richieste dal firmware del bracciale sono state sviluppate e collaudate con la stessa metodologia.

Il protocollo di comunicazione e la relativa libreria sono stati testati visionando direttamente il messaggio processato utilizzando la seriale hardware disponibile in Fishino UNO e il Monitor seriale disponibile in Arduino IDE.

Per quanto riguarda la comunicazione tra il modulo HM-10 e l'applicazione smartphone Android è stata testata e sviluppata utilizzando un'applicazione di terze parti (https://github.com/googlesamples/android-BluetoothChat) da cui è stato anche ricavato il codice di gestione del Bluetooth Low Energy per l'applicazione Lewe2.0.

Lo sviluppo dell'applicazione è stato modulare sviluppando e testando ogni componente prima di assemblarli insieme. Per il debugging è stata sviluppata una classe (Logger) con la quale è possibile stampare delle stringhe di debug (e di errore) utilizzando lo strumento lo strumento LogCat contenuto in Android Studio.

CONCLUSIONI E LAVORO FUTURO

Il progetto Lewe2.0 ha visto la realizzazione di un primo prototipo di un protocollo di comunicazione nell'ambito IOT che cerca di rispettare alcuni requisiti fondamentali che sono leggerezza, flessibilità, efficienza e sicurezza.

Il prototipo ha evidenziato che il formato JSON non è adeguato a questo tipo di applicazioni in quanto la sua intrinseca lunghezza dei messaggi degrada notevolmente le prestazioni e risorse di device che, per risparmiare energia, sono dotati del minino necessario per l'esecuzione del processo a loro assegnato.

Altresì un formato interamente binario risulta sicuramente una scelta migliore per questi contesti applicativi, limitando però la comprensione umana dei messaggi scambiati tra i vari host e rendendo di conseguenza difficoltoso il processo di debugging.

Probabilmente un formato per questo protocollo dovrà essere intermedio tra il binario e il JSON ricucendo, in questo modo, le risorse necessarie all'elaborazione senza degradare troppo la leggibilità umana degli stessi.

Un altro aspetto che è stato evidenziato da questo progetto è la necessità di ampliare le funzionalità del protocollo di comunicazione inserendo alcune tipologia di messaggi per poter fornire supporto a diversi contenti applicativi differenti da quello preso in esame.

Alcune di queste integrazioni possono essere l'inserimento di messaggi la cui bassa rilevanza non necessita conferma da parte dell'host destinatario o ancora una speciale tipologia di messaggi che dia la possibilità ai device di indicare alla rete la loro presenza e le funzionalità messe a disposizione della stessa.

Tutto il materiale presentato in questo documento è stato posto sotto licenza Apache 2 ed è fruibile gratuitamente ai seguenti indirizzi:

Applicazione per smartphone Android
https://github.com/alessandro1105/Lewe2.0App

Firmware e schemi per il bracciale
https://github.com/alessandro1105/Lewe2.0

Libreria HashMap (Arduino IDE)
https://github.com/alessandro1105/HashMap_Arduino_Library

Implementazione del protocollo Jack (Arduino IDE)
https://github.com/alessandro1105/Jack_Arduino_Library

Libreria SoftwareSerialJack per il protocollo Jack (Arduino IDE)
https://github.com/alessandro1105/SoftwareSerialJack_Arduino_Library

Tutte le componenti del progetto sviluppate da terzi sono di proprietà dei rispettivi sviluppatori ed è quindi necessario fare riferimento a loro per qualsiasi concessione sull'utilizzo di tale materiale.