Ricordate il progetto **Fish'n Tweets**, che permetteva di controllare gli I/O di Fishino tramite dei tweet? Ebbene, si è trattato del nostro primo esperimento di interazione tra Fishino e il mondo dei Social. L'applicativo è tutt'ora valido ma, a causa della "lentezza" di Twitter, a volte il tempo intercorso tra un comando e la sua esecuzione può superare i 20 secondi. Twitter inoltre è piuttosto complesso da gestire, soprattutto nella fase iniziale di creazione delle chiavi di accesso, dell'utente "robot", eccetera. Visto che volevamo tornare sull'argomento e che San Valentino incombe, abbiamo pensato di prendere i classici "due piccioni con una fava" realizzando un'applicazione, -stavolta basata sul servizio di instant messaging Telegram- che può essere un modo suggestivo per mandare messaggi personalizzati alla propria amata, stampati da una micro-stampante. I messaggi possono essere sia personalizzati che scelti da un database predisposto su un server. L'applicazione qui proposta non è limitata al giorno degli innamorati ma trova impiego nella realtà quotidiana, perché implementa anche una funzione di gestione della "lista della spesa", che permette di memorizzare una sequenza di prodotti, inviandone il nome tramite un messaggino Telegram in qualsiasi momento e permettendo di consultarla e/o stamparla, sempre da remoto. Ciò trova applicazione ad esempio in un ristorante, dove l'avventore può fare il proprio ordine, che verrà stampato alla cassa in automatico. L'applicazione ci permette di introdurre una libreria di interfaccia con Telegram appositamente sviluppata, che ci consentirà non solo di controllare Fishino da qualsiasi parte del mondo, ma anche di leggerne gli ingressi e, se vogliamo, di farci avvisare in caso si verifichino eventi particolari, sempre tramite un messaggio Telegram. La libreria è dotata dei necessari strumenti di "sicurezza": è possibile lasciare l'accesso a tutti, vincolarlo solo ad utenti specifici oppure, se necessario, lasciarlo aperto a tutti salvo alcuni utenti "disturbatori". Come vedrete la libreria si presenta in modo piuttosto modulare e semplice da usare, nascondendo all'utente i dettagli tecnici dell'implementazione e della comunicazione con Fishino. Questi applicativi possono funzionare anche con una scheda Arduino dotata di shield WiFi, tuttavia dipende dallo shield. Quello più comunemente disponibile (ed economico) non supporta la connessione sicura (SSL/HTTPS), quindi non è utilizzabile per l'interfacciamento con i social che la richiedono, come Twitter, Telegram ed altri. Disponendo, invece, di uno shield più recente, è possibile adattare la libreria ad esso.

TELEGRAM E I SUOI "BOT"

Cos'è un "bot"? In sintesi, è un account "virtuale" di Telegram,

NOTES MACHINE CON FISHINO

di MASSIMO DEL FEDELE

Appoggiandoci a un bot, stampiamo con una stampantina termica remota messaggi composti su Telegram. Una soluzione dalle mille applicazioni, utilizzabile per stampare liste e ordini in un locale e, perché no, per mandare messaggi stampati alla propria "bella" il giorno di San Valentino e non solo.

per creare il quale non servono numeri di telefono, dati personali, eccetera. Questo tipo di account deve essere creato da un account "reale"; per il resto, può funzionare come un utente simulato, ricevere ed inviare messaggi, ed altro.
A noi interessa qualcosa che possa interagire sia con il nostro Fishino che con uno o più utenti; creeremo quindi un bot che verrà controllato da Fishino e che sarà in grado di rispondere alle nostre richieste e/o inviarci messaggi.
Il bot è quindi un robot, programmabile per eseguire determinate azioni. I bot:

- non mostrano il loro stato (online/offline); sono sempre attivi e "vedono" all'istante (o quasi) i nostri messaggi;
- hanno una memoria limitata; i vecchi messaggi possono sparire dal server poco dopo essere stati processati;
- non possono iniziare una conversazione ma rispondono se interessati; per comunicare con un bot occorre aggiungerlo ad un gruppo oppure inviargli un messaggio iniziale;
- hanno un "nome utente" che finisce sempre in "bot"; per esempio, **TriviaBot**, **Fishino_bot**;
- anche se aggiunti a un gruppo, non ricevono tutti i messaggi se non impostati allo scopo.

CREIAMO IL NOSTRO BOT

Iniziamo subito con la creazione del bot che ci servirà per interagire con Fishino.
Per prima cosa, se non l'avete ancora fatto, occorre scaricare ed installare Telegram; nelle immagini in queste pagine vedete l'installazione su un desktop Linux (l'abbiamo fatto per catturare le schermate), ma normalmente l'applicazione è installata su uno smartphone. Nell'installazione vi verrà chiesto di confermare il vostro numero di telefono e/o un codice di sicurezza, a seconda della piattaforma. Una volta installato ed eseguito, vi troverete di fronte alla schermata di **Fig. 1**. Per creare il bot bisogna ricorrere al "padre di tutti i bot", ovvero all'utente BotFather: è sufficiente digitarne il nome nella casella di ricerca (**Fig. 2**). Vi apparirà la schermata introduttiva di BotFather; cliccate su **Start** (in basso) per avviare il bot; BotFather risponderà con un messaggio esplicativo contenente una lista dei principali comandi disponibili (**Fig. 3**).
Siccome vogliamo creare un nostro bot, clicchiamo sul comando **/newbot** (**Fig. 4**). BotFather ci chiede di dare un nome al nostro bot; scegliamo un nome a caso, per esempio **Pippo** (**Fig. 5**).
Ci viene quindi chiesto uno '**user name**' per il nostro bot; questo deve per forza essere unico e terminare in 'bot'; nel caso scegliessimo uno username già occupato BotFather ci avviserà e ci imporrà di cambiarlo. In questo caso PippoBot è già in uso (ovviamente!), quindi sceglieremo **PippoRobot** che casualmente è libero; BotFather ci avviserà

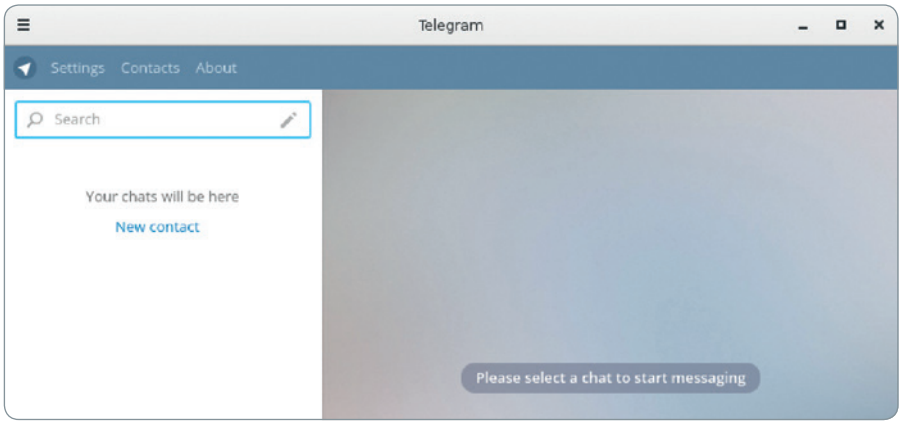


Fig. 1 *- Installazione di Telegram.*

dell'avvenuta creazione del nostro bot e ci darà le istruzioni per utilizzarlo da remoto (**Fig. 6**). Come potete notare, il nostro bot è accessibile tramite l'indirizzo Telegram:

telegram.me/PippoRobot

e per controllarlo è necessario possedere il token di sicurezza fornito da **BotFather**:

Use this token to access the HTTP API:
nnnnnnnnn:XXXXXXX-YYYYYYY_ZZZZZZZZZZZZZZZZZZZ

Il token (qui oscurato) è indispensabile per garantire la sicurezza che nessuno possa "manomettere" il nostro bot. Questa token andrà inserita nel codice del nostro sketch, come vedremo in seguito, quindi prendetene nota e, soprattutto, mantenetela ben segreta visto che chiunque ne viene in possesso è in grado di fare qualsiasi cosa con il nostro bot, compreso cancellarlo. Clicchiamo ora sul link *telegram.me/PippoRobot* (**Fig. 7**) e successivamente sul comando **Start**, che avvierà il nostro bot inviandogli un messaggio iniziale (**Fig. 8**). Ecco fatto! Abbiamo creato il nostro primo bot di Telegram!

LA LIBRERIA FISHGRAM

Una volta creato il nostro bellissimo bot occorre, perché sia utile, potergli inviare e ricevere messaggi, oltre che dall'app Telegram, anche attraverso il nostro Fishino.
Per interagire con il bot Telegram utilizza il protocollo HTTPS con un formato ben specifico; in particolare, i comandi possono essere impartiti tramite direttive HTTP GET e POST, e negli headers http inviati deve sempre essere presente la nostra token.
Per esempio, una richiesta di tipo '**getUpdates**', che ci permette di richiedere i messaggi ricevuti dal bot ha il formato seguente:

```
GET /botTOKEN/getUpdates?offset=
nnn&timeout=4&limit=1&allowed_
updates=messages HTTP/1.1
User-Agent: FishGram 1.0.0
Host: api.telegram.org
```

In questo formato di richiesta:

- **TOKEN** è la nostra token di accesso, vista al paragrafo precedente;
- **offset=nnnn** rappresenta il primo ID di messaggio cui siamo interessati;
- **limit=1** indica a Telegram di fornirci un solo messaggio per richiesta (Telegram può fornire più messaggi per ogni richiesta di aggiornamento);
- **allowed_updates=messages** indica che siamo interessati solo ai messaggi e non, per esempio, ai file audio e/o immagini.

Possiamo provare il comando

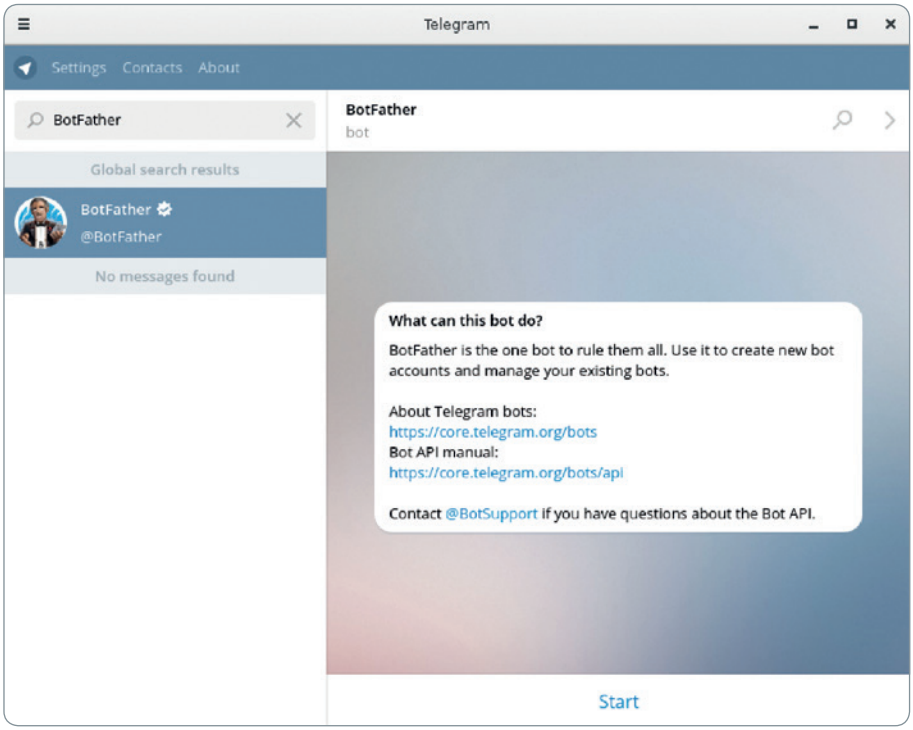


Fig. 2 *- Ricerca di BotFather.*

direttamente sul browser, immettendo nella casella dell'indirizzo il testo seguente (sostituite TOKEN con il vostro token di accesso):

https://api.telegram.org/botTOKEN/getUpdates?offset=0&timeout=4&limit=1&allowed_updates=messages

Scrivete, ovviamente, tutto su una sola riga. Telegram risponderà con un testo in formato JSON come il seguente:

```
{"ok":true,"result":[{"update_
id":904783368,
"message":{"message_id":2,"f
rom":{"id":nnnnnnnnn,"first_
name":"Massimo","last_name":"Del Fe
dele"},"chat":{"id":nnnnnnnnn,"fir
st_name":"Massimo","last_name":"Del
Fedele","type":"private"},"date":14
83380231,"text":"ciao Pippo"}}]}
```

Come si può notare, il testo di risposta contiene parecchie informazioni, tra cui il mittente del messaggio, l'id del messaggio (tramite il quale, per esempio, in una chiamata successiva inviando come **offset=id+1** potremo richiedere i messaggi a partire dal successivo), eccetera.

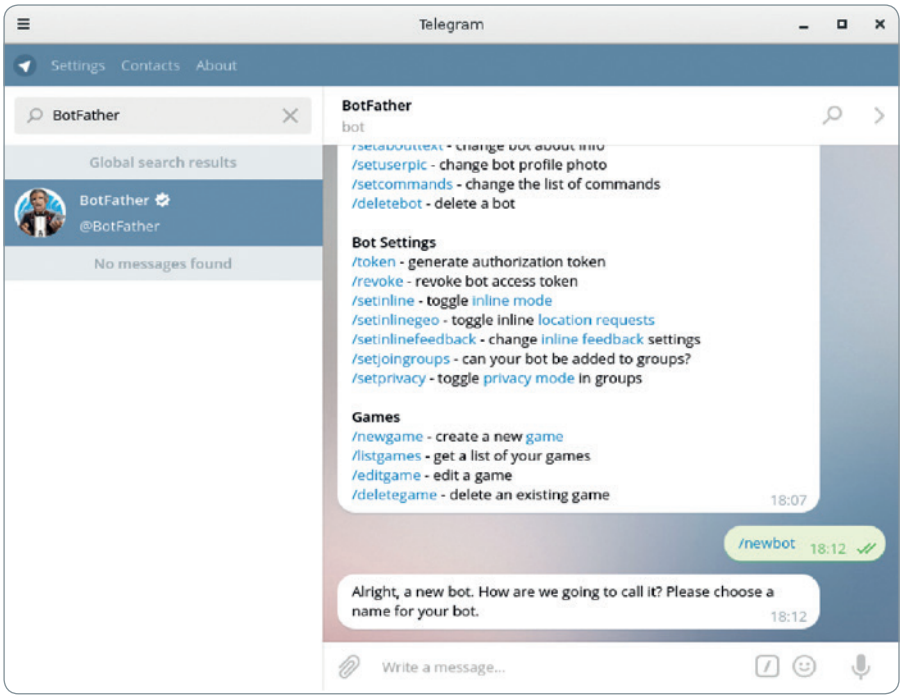


***Fig. 4** - Creazione del nostro Bot.*

È facile capire che, senza un'apposita libreria software, il funzionamento della cosa risulta piuttosto complesso. Se l'invio della richiesta è relativamente semplice (basta utilizzare un oggetto FishinoClient, collegarlo all'indirizzo richiesto, ed inviare una serie di stringhe di caratteri), l'analisi del JSON restituito non è altrettanto banale, soprattutto a causa delle risorse limitate del nostro Fishino, specialmente per quanto riguarda la memoria RAM.

Per questo ci viene in aiuto la libreria **JSONStreamingParser**, scritta per il progetto Fish'n Tweets, che è in grado di "digerire" l'output di Telegram, carattere per carattere, senza bisogno di memorizzare nulla, e di spezzettarlo in tanti elementi del tipo **nome: valore** e, ad ogni elemento ricevuto, chiamare una funzione da noi definita.

Siccome spiegare il funzionamento della libreria richiederebbe metà delle pagine di questo fasci-

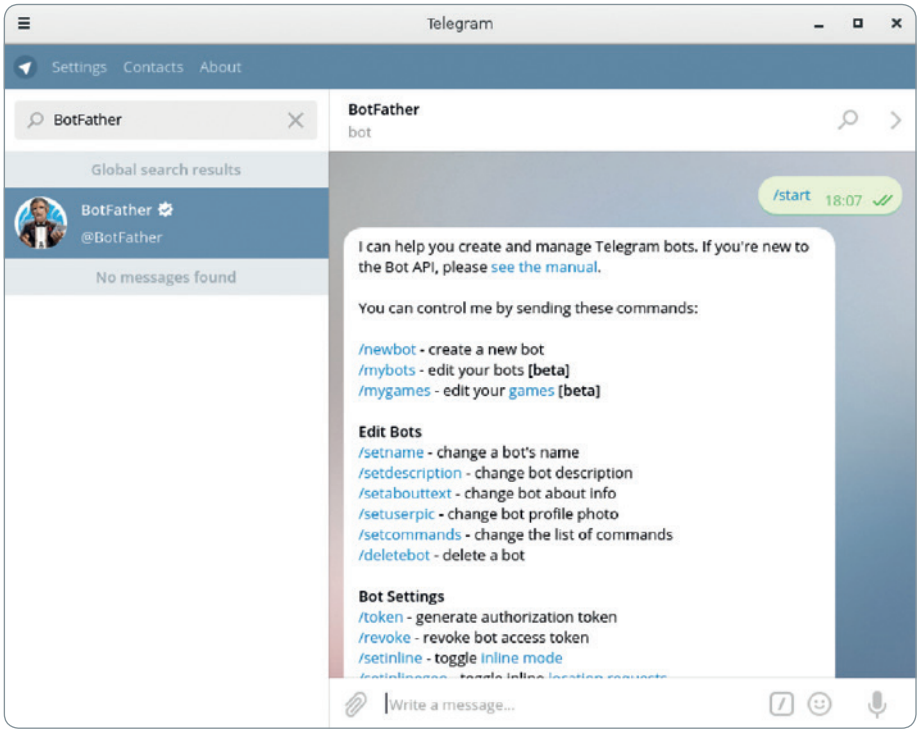


***Fig. 3** - Risposta di BotFather.*

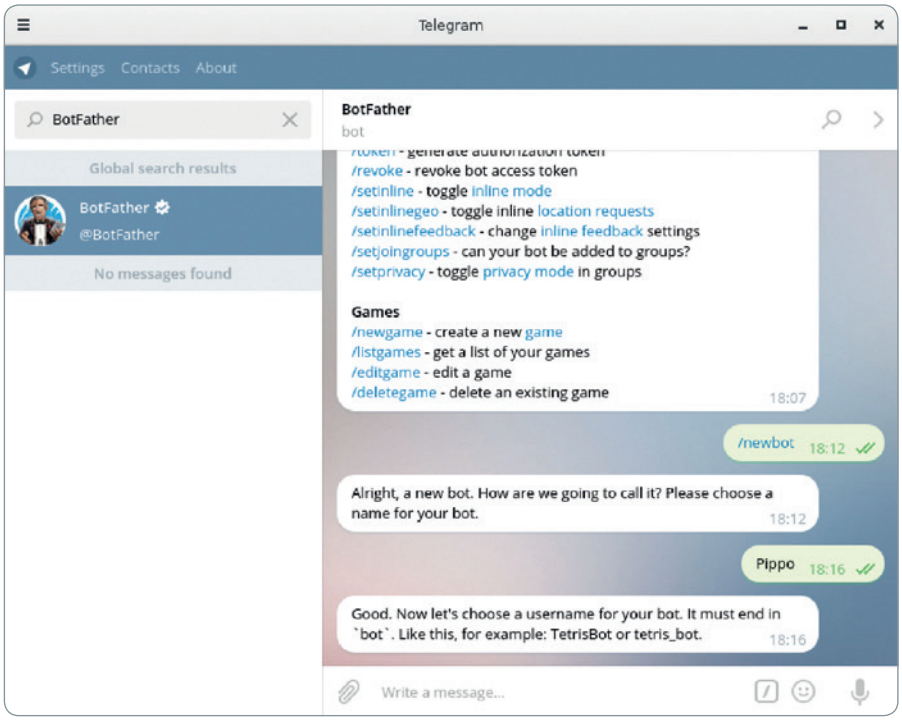


***Fig. 5** - Assegnazione del nome al Bot.*

quali è costituito dalla funzione di gestione degli eventi (di cui parleremo tra poco):

```
bool FishGramHandler(uint32_t id,
const char *firstName, const char *la-
stName, const char *message)
```

Il secondo è "inizializzazione di FishGram":

```
        // start FishGram
        FishGram.
event(FishGramHandler);
        FishGram.begin(F(MY_TELE-
GRAM_TOKEN));
```

E il terzo è la chiamata nel loop:

```
        FishGram.loop();
```

Semplice, vero? Ma come funziona? Ebbene, a differenza della maggior parte degli sketch cui siamo abituati, la libreria FishGram funziona per eventi; una volta inizializzata, resta quasi invisibile e, soprattutto, occupa

colo, se non oltre, partiremo da un piccolo sketch di esempio per mostrarne l'utilizzo; chi volesse approfondire troverà il codice della libreria stessa molto commentato e quindi relativamente facile da capire. Affronteremo comunque, durante la spiegazione dell'utilizzo, di alcune peculiarità e scelte apparentemente strane che abbiamo effettuato nella scrittura della stessa.

UN PRIMO SKETCH, SEMPLICE SEMPLICE

Iniziamo... dall'inizio! Proviamo a visualizzare i messaggi che vengono inviati al nostro bot sul monitor seriale; per far questo lo sketch è davvero semplicissimo. Per questo (primo) esempio, inseriremo il codice completo (compresa l'inizializzazione di Fishino, la connessione al router WiFi, eccetera, in modo da permettere a chiunque di collaudarlo "al volo" semplicemente copiando il codice nell'IDE; per i prossimi esempi eviteremo tutto questo ed inseriremo solo il codice specifico di **FishGram** (**Listato 1**).

Come potete vedere, il grosso dello sketch è l'inizializzazione di Fishino, il collegamento al router, eccetera. Per quanto riguarda FishGram in pratica troviamo rilevanti tre punti, il primo dei

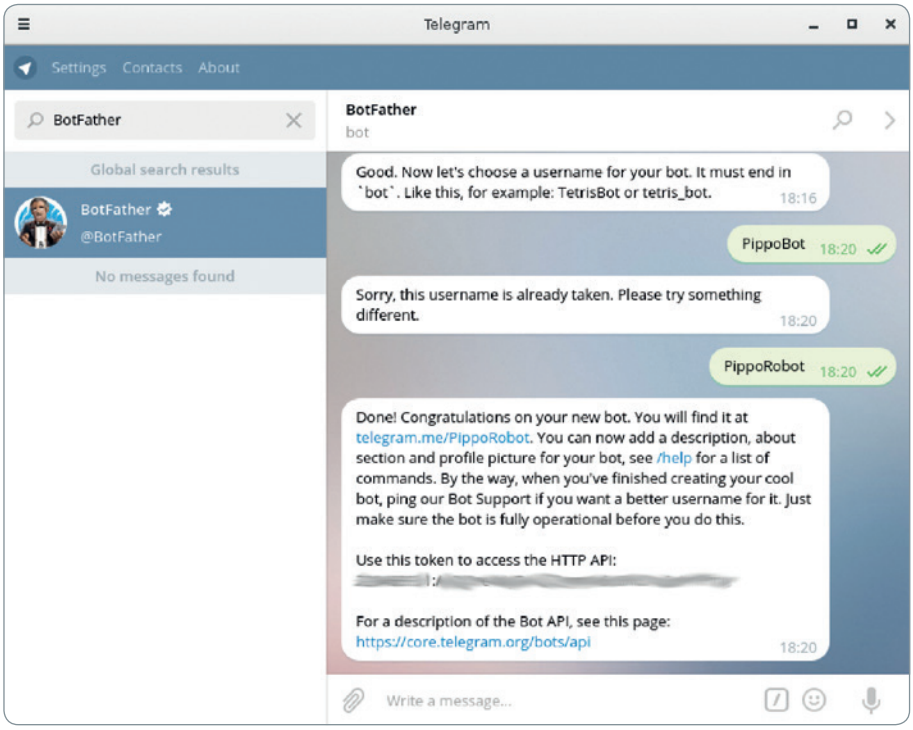


***Fig. 6** - Completamento del bot.*

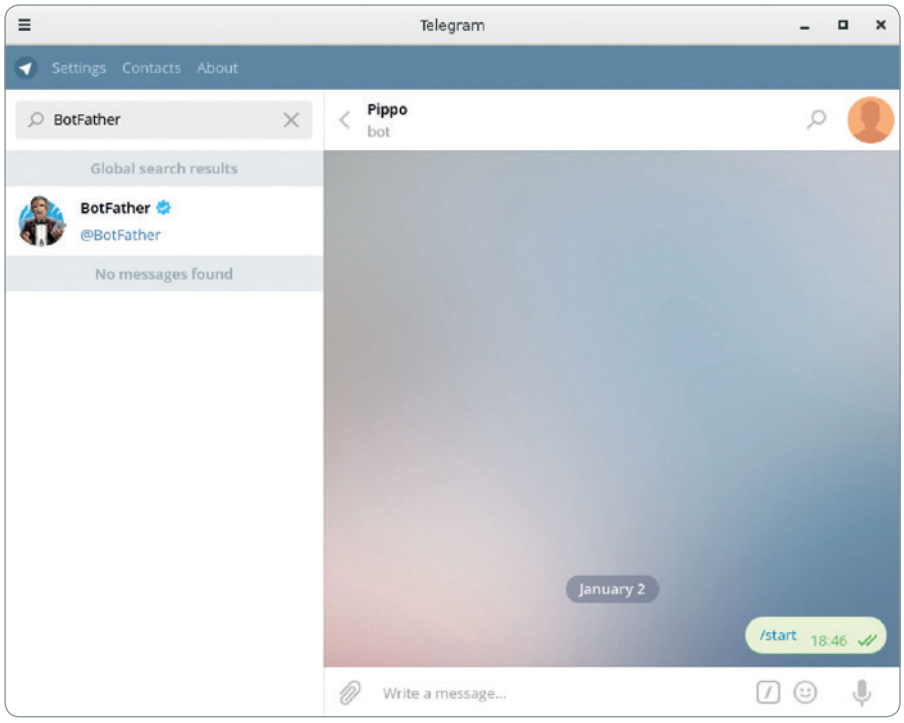


***Fig. 7** - Avvio del bot.*

pochissime risorse di calcolo e di memoria. L'importante è che la chiamata `FishGram.loop();` dentro al loop venga eseguita spesso.
Questa chiamata, che normalmente dura poche decine di microsecondi, consente alla libreria di gestire un timer interno e di richiamare, ad intervalli prefissati, il server di Telegram e richiedere eventuali aggiornamenti.
Se questi aggiornamenti non sono presenti, torna senza fare nulla; se invece c'è qualche novità la raccoglie, organizza e chiama la funzione evento `FishGramHandler()` passandole come parametri l'id, il nome ed il cognome del mittente ed il testo del messaggio.
Con questa funzione possiamo quindi stampare sulla porta seriale quello che riceviamo.
Le due chiamate nel setup, ovvero la `FishGram.event()` e la `FishGram.begin()`, servono rispettivamente per collegare la nostra funzione di gestione degli eventi e per dire a FishGram chi è il nostro bot tramite il token.
Non saremo quindi noi, dentro allo sketch, a doverci ricordare di chiedere a FishGram gli aggiornamenti "ogni tanto", ma con questo meccanismo sarà la stessa libreria ad avvisarci se e solo se arriva qualcosa di interessante dal nostro bot.
Quali sono gli svantaggi di un simile metodo? In pratica ce n'è solo uno: se la `FishGram.loop()` non viene eseguita abbastanza spesso il programma risponde male. Per esempio, se infilassimo una **delay(1000)** dentro alla **loop()**, la libreria riuscirebbe ad elaborare solo un carattere ricevuto ogni secondo, quindi passerebbero giornate prima di avere una risposta, o, più probabilmente, il server di Telegram chiuderebbe la connessione prima.
Quindi è ben possibile (anzi, lo scopo di questo sistema è proprio quello!) far eseguire al nostro Fishino qualcos'altro, mentre è in attesa di messaggi, ma questo qualcos'altro non deve nè bloccare l'esecuzione del **loop()** nè impiegare troppo tempo. Quindi, niente **delay()**, niente **while()** con durate lunghe e/o attesa di pulsanti da premere, eccetera. Tutto

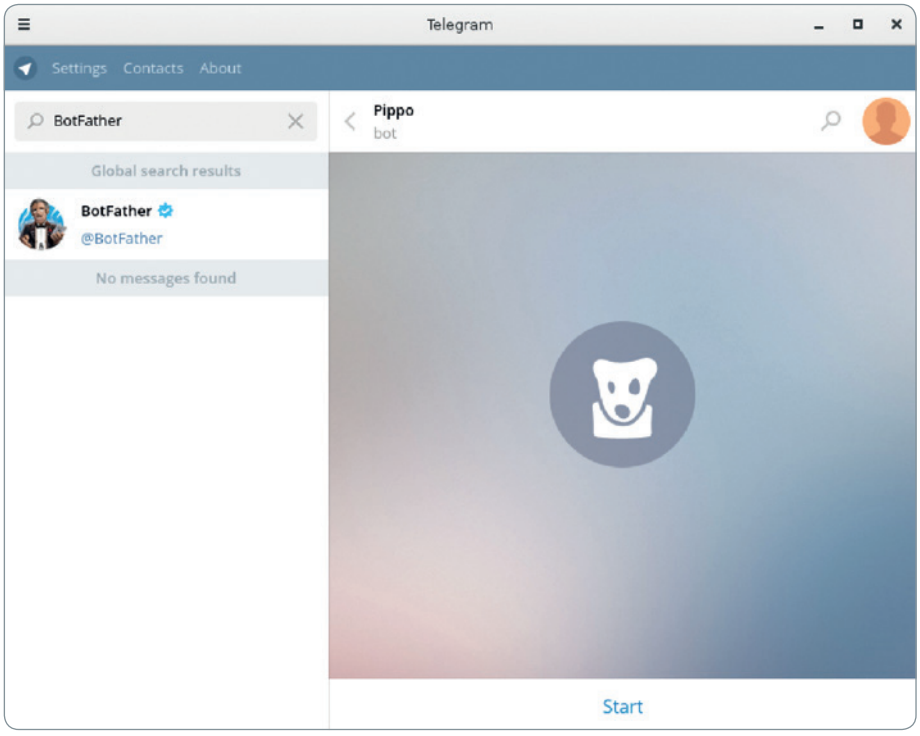


***Fig. 8** - Messaggio iniziale del bot.*

Listato 1

```
#include <Fishino.h>
#include <JSONStreamingParser.h>
#include <FishGram.h>
#include <SPI.h>

#define MY_SSID     "IL_MIO_SSID"
#define MY_PASS     "LA_MIA_PASSWORD"

// se volete un IP dinamico basta commentare/eliminare le 3 righe seguenti
#define IPADDR      192, 168,  1, 251
#define GATEWAY     192, 168,  1, 1
#define NETMASK     255, 255, 255, 0

#define MY_TELEGRAM_TOKEN "LA_TOKEN_DEL_MIO_BOT_TELEGRAM"

#ifdef IPADDR
        IPAddress ip(IPADDR);
        #ifdef GATEWAY
                IPAddress gw(GATEWAY);
        #else
                IPAddress gw(ip[0], ip[1], ip[2], 1);
        #endif
        #ifdef NETMASK
                IPAddress nm(NETMASK);
        #else
                IPAddress nm(255, 255, 255, 0);
        #endif
#endif

// fishgram event handler -- just display message on serial port
bool FishGramHandler(uint32_t id, const char *firstName, const char *lastName, const char *message)
{
        Serial << F("Ho ricevuto un messaggio da '") << firstName << "'\n";
        Serial << F("Il messaggio dice: '") << message << "'\n";
        return true;
}

void setup(void)
{
        Serial.begin(115200);
        while (!Serial)
                ;

        while(!Fishino.reset())
                Serial << F("Fishino RESET FAILED, RETRYING...\n");
        Serial << F("Fishino WiFi RESET OK\n");

        Fishino.setMode(STATION_MODE);
        Fishino.setPhyMode(PHY_MODE_11G);

        Serial << F("Connecting to AP...");
        while(!Fishino.begin(MY_SSID, MY_PASS))
        {
                Serial << ".";
                delay(2000);
        }
        Serial << "OK\n";

#ifdef IPADDR
        Fishino.config(ip, gw, nm);
#else
        Fishino.staStartDHCP();
#endif

        Serial << F("Waiting for IP...");
        while(Fishino.status() != STATION_GOT_IP)
        {
                Serial << ".";
                delay(500);
        }
        Serial << F("OK\n");

        // start FishGram
        FishGram.event(FishGramHandler);
        FishGram.begin(F(MY_TELEGRAM_TOKEN));
}

void loop(void)
{
        FishGram.loop();
}
```

dev'essere eseguito senza bloccare il loop, quindi con **millis()** per i ritardi e controlli "volanti" per eventuali pulsanti da rilevare. Esistono alternative ma vanificano in parte i vantaggi dell'approccio; ad esempio, se vogliamo attendere che un I/O vada a livello HIGH, invece di fare:

```
while(!digitalRead(5))
        ;
```

possiamo scrivere:

```
while(!digitalRead(5))
        FishGram.loop();
```

In modo che FishGram continui ad essere richiamata nell'attesa. Allo stesso modo, invece di utilizzare:

```
delay(1000);
```

dovremo procedere con l'utilizzo della **millis()** in questo modo:

```
uint32_t tim = millis() + 1000;
while(millis() < tim)
        FishGram.loop();
```

Ma è possibile aggiungere alla libreria una funzione del tipo seguente?

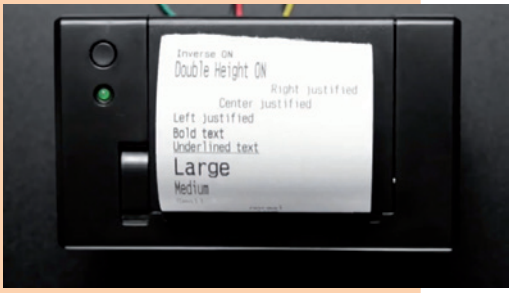
```
FishGram.aspettaMessaggio()
```

Certamente si... ma se il messaggio non arriva, lo sketch rimane bloccato all'infinito oppure per un tempo prefissato senza comunque poter fare altro, quindi abbiamo appositamente evitato di introdurla.
Sul monitor seriale verrà visualizzato qualcosa di simile:

```
Fishino WiFi RESET OK
Connecting to AP...OK
Waiting for IP........OK

Ho ricevuto un messaggio da 'Massimo'
```

La **stampante termica**



Per stampare i nostri messaggi inviati da Telegram ci siamo avvalsi di una piccola stampante alfanumerica (comunque stampa anche i caratteri del cinese) a carta termica facilmente reperibile in commercio e negli store on-line per maker, come Adafruit e Sparkfun.
La stampantina è del tipo a 20 colonne e stampa su rotolo di carta termica da 57,5 ±0,5 mm e si interfaccia tramite una connessione seriale a livello TTL. Di seguito ne elenchiamo le caratteristiche.

- tensione di alimentazione: 5÷9 Vcc;
- assorbimento max: 1,5 A;
- velocità di stampa: 50÷80 mm/s
- risoluzione: 8 pixel/mm, 384 pixel/linea
- larghezza effettiva di stampa: 48mm
- set caratteri: ASCII,GB2312-80 (cinese);
- font stampa: ANK:5×7, cinese: 12x24,24×24;
- protocol: TTL Serial, 19.200 baud;
- dimensioni (LxPxH): 111x65x57 mm;
- temperatura di esercizio: 5°÷ 50°C.

La stampante è la classica termica simile a molti modelli esistenti da anni, però nasce -guardacaso- per il mondo Arduino; allo scopo viene fornita con a corredo l'apposita libreria firmware. Nello specifico, nel nostro sketch per Fishino abbiamo integrato e utilizzato la stessa libreria fornita da Adafruit.

```
Il messaggio dice: 'Ciao Pippo,
come stai?'
```

INVIARE UN MESSAGGIO DA FISHINO A UN UTENTE

Se ricordate, nel paragrafo sulle differenze tra un bot ed un umano era specificato un punto apparentemente poco importante ma che invece è fondamentale: un bot non può iniziare una conversazione. Non possiamo quindi dire al nostro Fishino "cerca Giuseppe e spediscigli un messaggio". Questa è stata una scelta ben precisa di chi ha sviluppato le API di Telegram, per impedire un uso improprio dei bot. Come potete ben immaginare, se ci fosse questa possibilità sarebbe semplicissimo realizzare uno spam-bot in grado di inviare migliaia di messaggi indesiderati a chiunque! Come possiamo fare, quindi? Semplicemente, una volta inviato almeno un messaggio al nostro bot, questo avrà a disposizione l'id della chat (che poi è anche l'id di chi ha inviato il messaggio) e potrà quindi rispondere a questo utente e ad altri eventuali che l'hanno contattato. Vediamo quindi una piccola modifica allo sketch, che ci permette di ottenere un feedback da parte di FishGram sul nostro telefonino; per far questo è sufficiente modificare la funzione di gestione degli eventi come nel **Listato 2**. E, al prossimo messaggio inviato, il nostro bot, oltre a mostrarci l'output sul monitor seriale, ci risponderà come mostrato in **Fig. 9**.
La funzione **FishGram.sendMessage()**, non dovendo attendere alcunché, si comporta come siamo abituati: viene richiamata dove vogliamo, esegue il suo compito e poi ci lascia proseguire.
Il primo parametro, id, è l'id dell'utente a cui si vuole inviare il messaggio; il secondo è il messaggio vero e proprio.

Il sistema in versione San Valentino: la stampante è montata in un quadro in tema.

LA LIBRERIA IN DETTAGLIO

Come avete visto nei due esempi precedenti, la libreria è piuttosto semplice; contiene comunque altre utili funzioni che vediamo qui di seguito. La prima è:

```
// end - termina la libreria
//FishGram
bool end(void);
```

e in pratica non serve mai, a meno di non utilizzare FishGram sporadicamente e voler liberare tutta la memoria che occupa.

```
// cancella i dati locali
FishGramClass &clear(void);
```

in pratica non è indispensabile; libera la memoria utilizzata da FishGram fino al prossimo evento.

```
// abilita/disabilita la lista di
//utenti ammessi
FishGramClass &restrict(bool b =
true);
FishGramClass &noRestrict(void) {
return restrict(false); }
```

queste ultime due funzioni

sono utilissime per eseguire un controllo su chi può mandare messaggi al nostro bot; se attivata con **restrict()** la funzione scarta automaticamente tutti i messaggi provenienti da id non contenuti nella lista al punto seguente.

```
// aggiunge un ID utente alla lista
//di utenti ammessi
FishGramClass &allow(uint32_t id);
```

questa funzione permette di aggiungere un id utente alla lista degli utenti ammessi; se la funzione viene abilitata con **restrict()**, solo gli utenti presenti in questa lista vedranno accolti i loro messaggi dal bot.

```
// aggiunge un ID utente alla lista
//di utenti bloccati
FishGramClass &block(uint32_t id);
```

Nel **Listato 3** invece vediamo la lista degli utenti bloccati; qualsiasi id presente in questa lista viene considerato come spam e quindi ignorato. All'avvio, di default FishGram legge l'ultimo messaggio disponibile, lo scarta ed attende nuovi messaggi; a volte è desiderabile leggere anche messaggi precedenti all'avvio, cosa ottenibile con questa funzione. Se introduciamo **recoverOldMessages(10)**, ad esempio, all'avvio, FishGram recupererà i 10 messaggi precedenti:

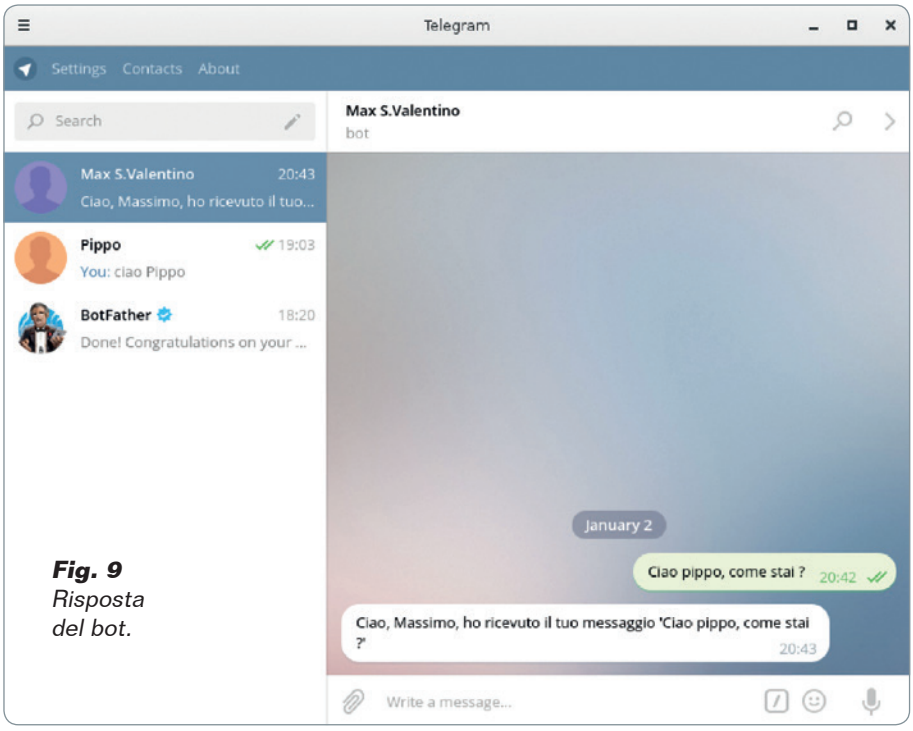


Fig. 9 *Risposta del bot.*

```
// imposta la funzione di gestione
//degli eventi
FishGramClass &event(FishGramEvent
e);
```

Questa funzione installa il gestore di eventi, come visto in precedenza:

```
// invia un messaggio ad un ID
//specifico
bool sendMessage(uint32_t const &id,
const char *msg);
bool sendMessage(uint32_t const &id,
const __FlashStringHelper *msg);
```

Queste due funzioni permettono di inviare un messaggio tramite **FishGram** a un **ID** specifico:

```
// invia un messaggio ad un ID spe-
cifico pezzo per pezzo
bool startMessage(uint32_t const &id,
uint16_t len);
bool contMessage(const char *msg);
bool contMessage(const __FlashStrin-
gHelper *msg);
bool contMessage(char c);
bool endMessage(void);
```

queste cinque funzioni permettono di inviare un messaggio **pezzo per pezzo**, quindi comode se la memoria RAM è scarsa. L'unico "inghippo" è che è necessario **conoscere a priori la lunghezza complessiva del messaggio**, da passare come parametro alla **startMessage()**. Il messaggio può poi essere inviato tramite varie chiamate alla **contMessage()**, e terminato con la **endMessage()**.

Listato 2

```
// fishgram event handler -- display message on serial port and give feedback to sender
bool FishGramHandler(uint32_t id, const char *firstName, const char *lastName, const char *message)
{
        Serial << F("Ho ricevuto un messaggio da '") << firstName << "'\n";
        Serial << F("Il messaggio dice: '") << message << "'\n";

        String s;
        s = "Ciao, ";
        s += firstName;
        s += ", ho ricevuto il tuo messaggio '";
        s += message;
        s += "'";
        FishGram.sendMessage(id, s.c_str());
        return true;
}
```

LA NOTE MACHINE

Dopo aver esaminato la libreria **FishGram** in dettaglio possiamo vederne un'applicazione un po'

più complessa. In principio le cose sono abbastanza semplici: basta "trasformare" i messaggi in comandi per il nostro Fishino, eseguire questi comandi ed inviare un'eventuale risposta/conferma al mittente. Come dicevamo nell'introduzione, abbiamo pensato di implementare un sistema che permetta di:

- stampare messaggi inviati da Telegram;
- rispondere ad un messaggio particolare con una citazione o un messaggino romantico;
- stampare una citazione o un messaggino romantico;
- gestire una lista di oggetti che potrebbe corrispondere ad una lista della spesa;
- consultare e stampare la stessa.

Scendendo in dettaglio, i comandi sono implementati analizzando l'inizio del messaggio inviato, ricercandovi le seguenti parole:

- **stampa** invia alla stampante il resto del testo del messaggio;
- **romantico** preleva una frase romantica da un database e la invia come risposta;
- **citazione** preleva una citazione da un database e la invia come risposta;
- **stampa romantico** preleva una frase romantica da un database e la stampa;
- **stampa citazione** preleva una citazione da un database e la stampa;
- **aggiungi** aggiunge un oggetto alla lista della spesa;
- **rimuovi** rimuove un oggetto dalla lista della spesa;
- **mostra lista** invia come risposta il contenuto della lista della spesa;
- **stampa lista** stampa la lista della spesa;
- **svuota lista** svuota la lista della spesa;
- **help** mostra l'elenco dei comandi disponibili (questa lista).

Listato 3

```
// imposta il numero di messaggi precedenti da recuperare
// -1 per TUTTI (usare con cautela!), 0 per nessuno.
// dev'essere chiamata prima della begin()
FishGramClass &recoverOldMessages(uint32_t n = (uint32_t)-1);
```

Ad esempio, inviando il messaggio '**stampa Ci vediamo stasera?**' la frase '**Ci vediamo stasera?**' verrà stampata immediatamente e, nello stesso tempo, riceveremo su Telegram un messaggio di conferma.
Vediamo quindi le parti salienti dell'applicazione. Innanzitutto, abbiamo bisogno di una funzione che gestisca l'evento proveniente da Telegram (quella che nei due piccoli esempi di prima si limitava a mostrare il messaggio sulla seriale e/o reinviarlo al mittente come conferma). In questo caso le cose si complicano leggermente, dovendo analizzare il testo del messaggio per estrarne i comandi. Abbiamo poi bisogno della **lista di comandi**, memorizzata in qualche modo "comodo" e di una serie di funzioni che gestiscano i comandi stessi.
Il modo più semplice per implementare la cosa è quello di una "**linked list**", ovvero una **lista di elementi collegati** alla quale è possibile aggiungere o togliere (quest'ultima funzione qui non è utilizzata) degli elementi. Gli elementi stessi, poi, sono costituiti dal **testo del comando** abbinato alla **funzione che lo gestisce**.
Iniziamo con questi ultimi, dichiarati come nel **Listato 4**.
La **typedef** iniziale serve a fornire una **sintassi abbreviata** per gestire le funzioni che eseguiranno i nostri comandi. In poche parole, queste funzioni hanno come parametri l'**id**, **nome** e **cognome** del mittente ed il **corpo del messaggio** senza il nome del comando; ritornano un valore

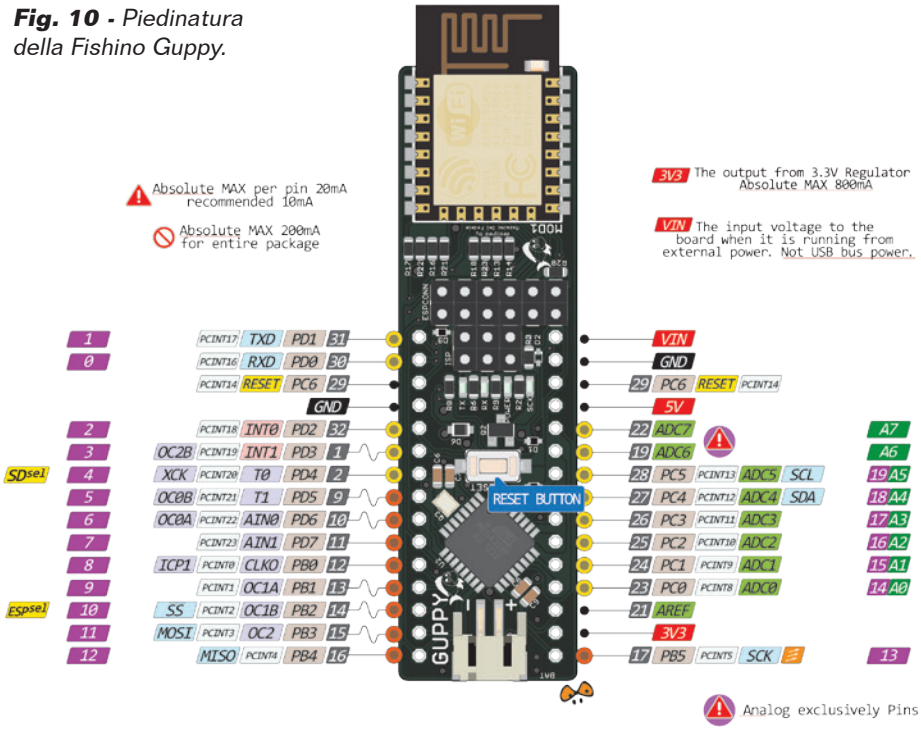


***Fig. 10** - Piedinatura della Fishino Guppy.*

Listato 4

```
typedef bool (*CommandHandler)(uint32_t, const char *, const char *, const char *);
struct CommandElement: public ListElement<CommandElement>
{
        const FLASH_HELPER *name;
        CommandHandler handler;
        CommandElement(const FLASH_HELPER *_name, CommandHandler _handler):
name(_name), handler(_handler) {}
};
```

Listato 5

```
commandList.add(new CommandElement(F("aggiungi")           , Cmd_AggiungiLista));
commandList.add(new CommandElement(F("rimuovi")            , Cmd_RimuoviLista));
commandList.add(new CommandElement(F("mostra lista")       , Cmd_MostraLista));
commandList.add(new CommandElement(F("stampa lista")       , Cmd_StampaLista));
commandList.add(new CommandElement(F("svuota lista")       , Cmd_SvuotaLista));
commandList.add(new CommandElement(F("stampa romantico")   , Cmd_StampaRomantico));
commandList.add(new CommandElement(F("romantico")          , Cmd_Romantico));
commandList.add(new CommandElement(F("stampa citazione")   , Cmd_StampaCitazione));
commandList.add(new CommandElement(F("citazione")          , Cmd_Citazione));
commandList.add(new CommandElement(F("stampa")             , Cmd_Stampa));
commandList.add(new CommandElement(F("help")               , Cmd_Help));
```

booleano in base all'esito del comando stesso (valore qui non usato, ma potrebbe servire per comandi differenti).
La **struct CommandElement** rappresenta invece la descrizione del comando vero e proprio, ovvero il testo (**name**) ed il puntatore alla funzione che lo gestisce (**handler**). Nella struct (che è praticamente la stessa cosa di una class, salvo qualche lieve differenza) abbiamo anche inserito un **costruttore** che permette di inizializzarla. La FLASH_HELPER è una macro (#define) che dipende dal controller usato; per gli 8 bit viene tradotta in **__FlashStringHelper**, per poter utilizzare le stringhe nella memoria flash (e quindi risparmiare preziosa RAM), mentre nei 32 bit viene tradotta in una semplice **char**, visto che per queste tipologie di controller non c'è differenza tra stringhe in RAM o nella Flash.
La lista dei comandi viene quindi rappresentata in questo modo:

```
List<CommandElement> commandList;
```

Qui utilizziamo una **template** scritta da noi per gestire una **lista di oggetti generici**. Perché una **template** (e soprattutto... cos'è una **template**???), e perché non utilizzarne una già fatta da altri? Alla prima domanda e mezza si può rispondere che una template è quello che si chiama una **classe generica**, ovvero può essere utilizzata per gestire differenti tipi di oggetti. Quindi, se volessimo realizzare una lista di patate, basterebbe scrivere:

```
List<Patate> listaPatate;
```

ed otterremmo quanto desiderato, senza dover scrivere una sola riga di codice aggiuntivo. Allo stesso modo, se desiderassimo una lista di Rose, potremmo scrivere:

```
List<Rose> listaRose;
```

Una bella differenza dal dover riscrivere ogni volta tutto il codice per gestire tipi di dati differenti!
La **template List** si trova nel file **List.h** dentro alla cartella del progetto; è probabile che in una futura versione delle librerie raccoglieremo un po' di codice riutilizzabile come questo in una libreria apposita. Analizzando il codice corrispondente, si nota che il template fornisce funzioni per aggiungere, eliminare e percorrere tutti gli elementi della lista; non si tratta di un codice particolarmente complesso nè performante, ma svolge egregiamente il suo compito.
Alla seconda domanda... non c'è una risposta: è vero che esistono in rete decine di implementazioni di Liste, Array, eccetera; semplicemente abbiamo preferito scriverne una limitata (e facilmente comprensibile) adatta al nostro

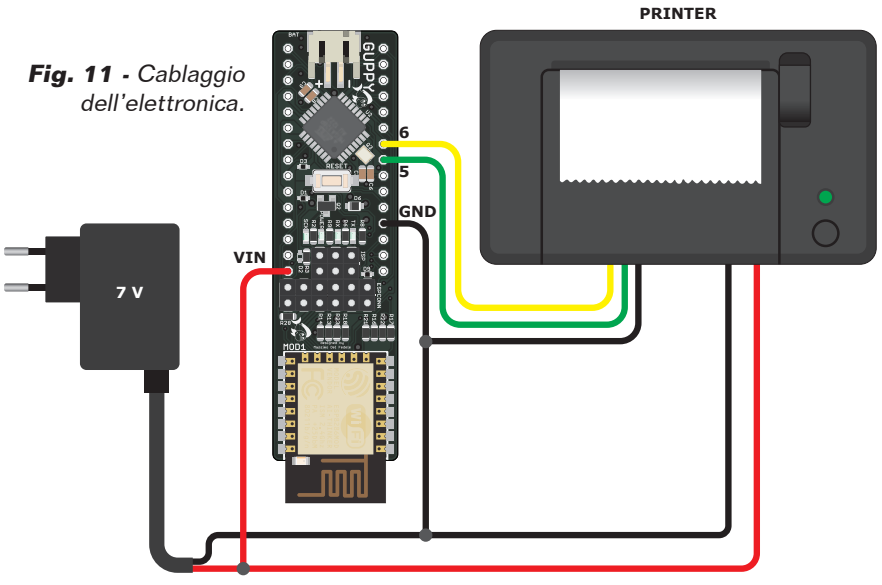


***Fig. 11** - Cablaggio dell'elettronica.*

scopo, anche per motivi didattici. Una volta dichiarata la lista di comandi occorre riempirla, e questo avviene nelle linee del **Listato 5**. La funzione **add** aggiunge un nuovo (**new**) elemento (**CommandElement**) inizializzandolo con il **nome** del comando e la funzione di gestione (**handler**) corrispondente. Ad esempio, la prima riga implementa il comando "**aggiungi**" gestito dalla funzione **Cmd_AggiungiLista()**.
Torniamo ora al gestore degli eventi di FishGram, che utilizza il codice di **Listato 6**.
Questo non fa altro che percorrere tutta la lista di comandi, confrontandoli con l'inizio del messaggio ricevuto; quando trova una corrispondenza termina la ricerca ed esegue il comando corrispondente.
Se non trova una corrispondenza invia al mittente un messaggio di errore (la penultima riga).
La funzione **starts()** richiamata dall'**handler** è un piccolo **helper** (funzione di utilità) che controlla se un testo inizia con una stringa prefissata, ed è definita poche righe sopra (**Listato 7**).
Non ci resta che vedere una delle funzioni di gestione dei comandi; per semplicità vedremo quella che implementa il comando 'stampa' (**Listato 8**).
Questa funzione non fa altro che stampare, come richiesto, il mesaggio (**sendToPrint(str)**) e risponderci con un messaggio di conferma, costruito per risparmiare ripetizioni con la helloMsg (che crea una stringa costituita da "Ciao <nome>, ", aggiungendovi il testo "ho stampato il tuo messaggio '", il testo del messaggio e la virgoletta di chiusura finale, reinviandolo poi al mittente tramite la FishGram.sendMessage().

CITAZIONI E LISTA DELLA SPESA

Per queste due funzionalità daremo solo una breve descrizione; non si tratta di codice particolarmente complesso ma appesantirebbe comunque troppo l'articolo. Il codice è comunque ben commentato e scaricabile dal nostro sito *www.elettronicain.it* e, in seguito, verrà incluso tra gli esempi nelle librerie di Fishino.
Le citazioni (e le frasi romantiche) vengono implementate per forza di cose appoggiandoci ad un server esterno, tramite un programmino in php contenente una serie di frasi preimpostate ed un algoritmo per poterne generare una casuale ad ogni accesso, a meno di non richiederne una specifica. Ma perché un php esterno? Non si potevano implementare

Listato 6

```
// fishgram event handler
bool FishGramHandler(uint32_t id, const char *firstName, const char *lastName, const char *message)
{
        CommandElement *elem = commandList.head();
        while(elem)
        {
                if(starts(message, elem->name))
                {
                        message += strlen_P((const char *)elem->name);
                        while(*message && isspace(*message))
                                message++;
                        return elem->handler(id, firstName, lastName, message);
                }
                elem = elem->next();
        }

        // command not found
        FishGram.sendMessage(id, F("Comando sconosciuto - inviare 'help' per lista comandi"));
        return false;
}
```

Listato 7

```
bool starts(const char *msg, const FLASH_HELPER *cmd)
{
        char c;
        uint16_t i = 0;
        while( (c = charAt(cmd, i++)) != 0)
                if(toupper(*msg++) != toupper(c))
                        return false;
        return true;
}
```

Listato 8

```
bool Cmd_Stampa(uint32_t id, const char *firstName, const char *lastName, const char *str)
{
        // print the message
        sendToPrint(str);

        // send a confirmation back to bot
        String ans = helloMsg(firstName, lastName);
        ans += F("ho stampato il tuo messaggio '");
        ans += str;
        ans += "'";
        FishGram.sendMessage(id, ans.c_str());
        return false;
}
```

Listato 9

```
// la lista della spesa
struct ShoppingListElement:public ListElement<ShoppingListElement>
{
        char *item;
        ShoppingListElement(const char *s) { item = strdup(s); }
        virtual ~ShoppingListElement() { if(item) free(item); }
};
List<ShoppingListElement>shoppingList;
```

direttamente sul Fishino, magari con l'utilizzo di una scheda SD? Si, ma non con le versioni **UNO** e **GUPPY** che abbiamo scelto per l'applicazione, soprattutto per motivi di compattezza. Queste infatti non hanno lo spazio di memoria sufficiente per gestire sia **FishGram** che la **scheda SD**. Utilizzando lo (scarso) spazio lasciato libero in Flash dall'applicazione avremmo potuto mettere insieme poche frasi, cosa che avrebbe portato a ripetizioni nel breve termine. Utilizzando una Fishino Mega o, ancora meglio, la nuova Fishino32 questi problemi spariscono completamente ed è possibile implementare in locale anche un grosso database di citazioni su scheda SD o anche direttamente nella memoria Flash del controller.
Tornando alle citazioni, la richiesta avviene, tramite protocollo HTTP (per i limiti di Fishino di non poter aprire più di una connessione HTTPS alla volta) tramite una semplice richiesta **GET** ad una path situata sul sito *www.fishino.it*. Il modulo PHP risponde con una stringa di testo semplice con questo formato:

```
ID, LEN, citazione
```

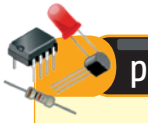
dove **ID** è un numero identificativo della citazione, per poterla "ripescare", ad esempio, per mandare al mittente la conferma di quanto si è stampato; **LEN** è la lunghezza del testo della citazione stessa, necessaria per poter utilizzare le funzioni **startMessage()** e annesse della libreria FishGram (ricordate? Permettono di inviare un messaggio anche carattere per carattere, senza quindi la necessità di doverlo scaricare per intero sul Fishino).
Il codice di richiesta della citazione al server contenuto nei moduli **Cit.h** e **Cit.cpp**, anch'essi reperibili nella cartella dell'applicazione, ed è di facile comprensione e ben commentato. Per quanto riguarda la lista della spesa, invece... abbiamo sfruttato ancora una volta la nostra **template List**, essendo questa perfetta per gestire questo tipo di dati; la dichiarazione è riportata sul **Listato 9**.
Come si può vedere, abbiamo prima definito una nuova struct contenente il dato che ci interessa (un semplice puntatore a carattere, che conterrà una stringa di testo allocata dinamicamente tramite strdup()); la novità qui è il distruttore (**~ShoppingListElement()**) che si occupa di liberare la memoria dinamica quando eliminiamo l'elemento.
La dichiarazione della lista è quindi immediata, come si vede dalla linea successiva alla struct, ed il suo utilizzo identico a quello relativo alla lista di comandi.
In questo caso utilizziamo anche la funzione remove() per eliminare, a richiesta, elementi dalla lista.
Per esempio, se vogliamo aggiungere "Pasta" alla lista tramite codice, possiamo scrivere:

```
shoppingList.add(new
ShoppingListElement("Pasta"));
```

Come accennato, la nostra implementazione di **List** è ben lungi dall'essere completa; manca, per esempio, una funzione di ricerca, che sarebbe utile ma che avrebbe complicato il codice oppure limitato la generalità dello stesso. Per cercare un oggetto nella lista dovremo quindi eseguirne una scansione completa "a mano", come fa il gestore di eventi di FishGram visto sopra.

L'ELETTRONICA

Il collegamento della stampante è davvero semplicissimo (**Fig. 11**); bastano tre cavetti: uno da collegare alla massa del Fishino, uno alla linea **D6** (il **TX**, ovvero l'**RX** della stampante) ed uno alla linea **D5** (l'**RX**, ovvero il **TX** della stampante). La stampantina comunica tramite un semplice protocollo seriale, per il quale sfruttiamo una **SoftwareSerial** tramite l'apposita libreria, ed è gestita dalla libreria **Adafruit_Thermal.**
Bene, si conclude qui la descrizione della nostra Note Machine, utilizzabile anche come spunto per controlli molto diversi, quali per esempio la gestione di luci, allarmi ed altro. Prossimamente presenteremo altre interessanti applicazioni di Telegram.



per il MATERIALE

Tutti i componenti utilizza-
t
f
c
s
r
u
r

Sostituire testo

sensore ad infrarossi IR38DM costa 2,50 Euro mentre l'inte-

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)
Tel: 0331-799775
http://www.futurashop.it