AREA DI PROGETTO 4D INFORMATICA
GRUPPO A

DOCUMENTAZIONE DEL PROTOCOLLO SERIALE E DEL SOFTWARE JAVA

ABBREVIAZIONI E DICITURE PARTICOLARI

-GUI = Graphical User Interface, indica nel nostro caso il programma scritto in Java;

-MICROCONTROLLORE = Scheda Arduino;

-Bit più significativo (MSB) = bit numero 7;

-Bit meno significativo (LSB) = bit numero 0;

-I bit sono numerati a partire da sinistra, considerando il loro peso nel sistema binario. Il MSB sarà quindi quello all'estrema sinistra mentre l'LSB sarà quello situato all'estrema destra;

-Java o applicazione, questi termini saranno frequentemente utilizzati per indicare l'applicazione in Java con la GUI che comunica con Arduino;

OBIETTIVO DEL DOCUMENTO

L'obiettivo principale del presente documento è illustrare il funzionamento del protocollo di comunicazione seriale e di come esso viene implementato all'interno del software Java.

Il protocollo sarà analizzato in tutte le sue parti e verranno fornite spiegazioni circa il suo funzionamento tramite tabelle e/o esempi.

Il software Java sarà analizzato in tutte le sue Classi e funzionalità, senza fornire esempi di codice, descrivendone le generalità e le motivazioni che hanno portato a determinate scelte di codifica.

Per quanto riguarda l'analisi del codice eseguito da Arduino, essa non è stata inclusa nella presente relazione in quanto il codice sorgente è stato prontamente accompagnato da commenti esaustivi direttamente nel file sorgente.

1. PROTOCOLLO DI COMUNICAZIONE

La scheda Arduino e l'applicazione Java (o GUI) comunicano tramite lo scambio di byte. Viene inviato sul canale seriale un messaggio composto da un byte, che è stato suddiviso, logicamente, in 8 bit. Ogni bit (o insieme di bit) rappresenta un'informazione comprensibile sia dal microcontrollore che dal software java.

La seguente tabella indica come verranno codificate le informazioni secondo il protocollo di comunicazione seriale:

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
L/S	*LED/MOT/SENS/AUTO*						
S	LED		ID LED*		OFF/ON/AUTO**		0
L	LED		ID LED*		AUTO 0/1	ON/OFF	0
S	MOTORINO		OFF/ON*	VALUE**			
L	MOTORINO		AUTO 0/1	ROT VALUE			
L	SENS		REALTIME VALUE%				
S	SENS		SET MINIMUM VALUE% (vedi tabella)				0
L	AUTO/SENS		GET MINIMUM VALUE% (vedi tabella)				0
S	AUTO		1	1	1	1	1

Si procede adesso all'analisi del significato di ciascun bit.

1.1. LETTURA E SCRITTURA

Il bit più significativo contenuto nel messaggio rappresenta la natura dell'istruzione: essa può essere di lettura o di scrittura.

Le diciture operazione di lettura o di scrittura sono relative esclusivamente all'operazione che deve essere svolta da Arduino. Un'operazione di scrittura comporterà una variazione delle proprietà della scheda o di un attuatore.

Un'operazione di lettura comporterà la comunicazione di una proprietà di Arduino all'esterno.

Le operazioni di scrittura vengono richieste esclusivamente dalla GUI: Arduino non invierà mai un byte di (scrittura).

Le operazioni di lettura sono gestite esclusivamente da Arduino, che comunicherà la variazione delle sue proprietà alla GUI non appena questa si verifichi: la GUI non invierà mai un byte di lettura.

Per questo motivo, possiamo intendere il bit rappresentante la lettura o la scrittura come l'identificativo del mittente:

0→Lettura→Mittente: Arduino

1→Scrittura→Mittente: Java

Arduino ignorerà tutti i byte che hanno come MSB lo 0, mentre Java ignorerà tutti i byte con MSB pari ad 1.

1.2. CODICE UNIVOCO PER OGNI COMPONENTE

Per capire a cosa fa riferimento il messaggio, ovvero su quale componente deve avvenire la scrittura/lettura, è stato assegnato un codice univoco ad ogni componente del sistema. I componenti sono 4: L'insieme dei LED, il Servo-Motore, il sensore di luminosità e la possibilità di attivare l'automazione. Per codificare questi componenti ci siamo serviti di due bit (in quanto 2^2=4=n° componenti) ed abbiamo assegnato ai led il codice 0_{10} $[00]_2$, al Servo-

Motore il codice 1_{10} $[01]_2$, al sensore di luminosità il codice 2_{10} $[10]_2$, ed all'automazione il codice 3_{10} $[11]_2$. Dopo aver definito le operazioni di lettura/scrittura e i codici dei vari componenti, possiamo "unire" i primi 3 bit ed ottenere il codice univoco per ogni operazione.

Per quanto riguarda gli altri bit, il significato a loro associato è strettamente dipendente dal tipo del componente. Nei prossimi punti verranno analizzati generalmente, mentre i dettagli saranno trattati in seguito.

1.3. MESSAGGIO ASSOCIATO AI LED (CODICI 000 E 100)

I bit 4 e 3 (numerati partendo dal bit meno significativo) rappresentano il codice univoco associato al singolo LED. Il presente protocollo, nella sua attuale configurazione, permette la codifica di non più di 4 LED (in quanto sono riservati alla loro codifica solo 2 bit). Questi due bit ci indicano precisamente su quale LED andare ad effettuare l'operazione di lettura o scrittura definita dai bit 2 ed 1.

Con i bit 2 ed 1 viene indicato lo stato del LED. Nel caso di un'operazione di scrittura (100), che riguarda un effettivo cambiamento di luminosità del LED, abbiamo 3 stati: spento (off), acceso (on) e automatico. Lo stato di spento viene indicato con il codice 0_{10} $[00]_2$, lo stato di acceso viene comunicato con il codice 1_{10} $[01]_2$, mentre per porre il LED in una configurazione automatica inviamo il codice 2_{10} $[10]_2$. Al codice 3_{10} $[11]_2$ non è associato alcun significato, pertanto i byte con tale codice verranno scartati da Arduino. Il bit meno significativo è ridondante, ovvero non è utilizzato: esso è generalmente posto a 0 ma il suo valore non concorre alla determinazione dell'informazione pertanto verrà ignorato dal microcontrollore.

Nel caso di un'operazione di lettura (000), che riguarda la comunicazione da parte di Arduino verso la GUI dell'effettivo valore di luminosità del LED, i bit 2 ed 1 possono dare vita a 4 codici utili ognuno rappresentante uno stato.

Possiamo, in questo caso, dividere gli stati in manuali ed automatici. Gli stati manuali indicano che il LED è soggetto al controllo dell'utente che lo ha acceso o spento fino a nuova comunicazione. Gli stati automatici indicano che il led è acceso o spento a seconda del valore di luminosità (e che quindi è attiva l'automazione).
Per differenziarli, li indicheremo con spento-manuale, acceso-manuale, spento-automatico e acceso-automatico.
Lo stato di spento-manuale viene indicato con il codice 0_{10} $[00]_2$, lo stato di acceso-manuale viene comunicato con il codice 1_{10} $[01]_2$, mentre per lo stato di spento-automatico viene indicato con il codice 2_{10} $[10]_2$ e per lo stato di acceso-automatico comunichiamo il codice 3_{10} $[11]_2$. Anche in questo caso l'LSB è ridondante e pertanto verrà ignorato.

1.4. MESSAGGIO ASSOCIATO AL SERVO MOTORE (CODICI 001 E 101)

Il bit 4 indica se il servo-motore è soggetto all'automazione controllata da Arduino, assumerà quindi il valore di 1 se l'automazione è attiva mentre assumerà il valore 0 se è disattivata.

Nel caso di un'operazione di scrittura, se il valore del quarto bit è pari ad 1, tutti i restanti bit (3 - 0) verranno ignorati in quanto l'informazione utile è soltanto quella di attivare l'automazione. Se parliamo di un valore manuale (bit 4 pari a 0) i restanti bit indicheranno il grado di rotazione del servo-motore. Potendo assumere un valore finito (da 0 a 15) è stato necessario creare una corrispondenza tra il valore desiderato, che oscilla tra gli 0 ed i 90 gradi e la rappresentazione in binario. Si è deciso di dividere per 6 il valore di rotazione (in quanto 90/6 = 15 = massimo rappresentabile), "sacrificando" tutti i valori non multipli di 6. In questo modo si ottengono 15 livelli di rotazione che sono selezionabili dall'utente. Questi valori sono sintetizzati nella seguente tabella:

Valore selezionato (mostrato GUI)	Valore INVIATO (decimale)	Byte inviato
0°	0	10100000
6°	1	10100001
12°	2	10100010
18°	3	10100011
24°	4	10100100
30°	5	10100101
36°	6	10100110
42°	7	10100111
48°	8	10101000
54°	9	10101001
60°	10	10101010
66°	11	10101011
72°	12	10101100
78°	13	10101101
84°	14	10101110
90°	15	10101111

Il microcontrollore dovrà semplicemente convertire il valore ricevuto e moltiplicarlo per 6 in modo da ottenere il valore di rotazione da applicare al servo-motore.

Per quanto riguarda le operazioni di lettura il procedimento è analogo: Arduino dopo aver letto il grado di rotazione del servo-motore, invierà alla GUI il valore diviso per 6 rappresentato dai bit 3-0. In questo caso il bit 4 non è vincolante ma rappresenta una pura informazione, in quanto l'obiettivo principale della lettura è inviare il valore della rotazione, sia esso causato da un automatismo o da altro.

1.5. MESSAGGIO ASSOCIATO AL SENSORE DI LUMINOSITA'

In questo caso i messaggi di scrittura e di lettura presentano differenze sostanziali: il messaggio di scrittura consiste nell'invio del

valore minimo di luminosità da considerare, mentre il messaggio di lettura serve a trasmettere il valore della luminosità ambientale attuale da Arduino verso Java.

1.5.1. MODIFICA DEL VALORE MINIMO (GUI→INO) (110)

La richiesta di modifica può essere inviata unicamente dalla GUI verso Arduino ed è codificata come operazione di scrittura.
Il valore inviato è compreso tra 0 e 10 ed il suo corrispettivo è stabilito dalla tabella delle corrispondenze (registrata anche nel codice Arduino).

BIT 7	BIT 6	BIT 5	BIT 4	BIT3	BIT2	BIT1	BIT0
S	SENS		VALORE [0; 10]				0
1	1	0	0000-1010 (1011-1111 non usati)				0

Il tutto è comprensibile tramite la seguente tabella delle corrispondenze:

Valore selezionato (mostrato GUI)	Valore INVIATO (decimale)	Byte inviato	Valore corrispondente Arduino
1%	0	11000000	49
10%	1	11000010	47
20%	2	11000100	45
30%	3	11000110	43
40%	4	11001000	41
50%	5	11001010	39
60%	6	11001100	35
70%	7	11001110	27
80%	8	11010000	17
90%	9	11010010	9
AUTO OFF	10	11010100	AUTO OFF

L'opzione 'AUTO OFF' disattiva per tutti i componenti il comportamento automatico (basato sulla luminosità) rendendo di fatto nulla la soglia minima di luminosità.

Come è stato possibile notare nella precedente tabella, i bit utilizzati sono 4-1 mentre il bit 0, ridondante, viene ignorato.
All'interno del codice di Arduino, la tabella è rappresentata da un array del tipo:

```
const float sensDb[10] = {49, 47, 45, 43, 41, 39, 35, 27, 17, 9};
```

dove l'indice corrisponde al valore ricevuto e il valore all'effettiva luminosità da impostare come minima.

I valori di luminosità sono stati scelti in relazione alla risposta reale del fotoresistore (collegato ai 3.3V) durante i test. In media, il valore massimo assunto dalla corrente in ingresso del fotoresistore, ponderato da una divisione della corrente in ingresso per 13, è di 50. Il valore, inizialmente, all'aumentare della luminosità, assume valori molto vicini (nel caso dei valori da 50 a 40), per poi scendere molto lentamente con valori sempre più distaccati, sino a giungere a 0 (nel caso di luminosità ambientale massima). Nonostante ciò, è molto raro ottenere una luminosità massima senza l'ausilio di una fonte luminosa direttamente adiacente al fotoresistore. In media i valori in un ambiente illuminato naturalmente oscillano tra i 50 ed i 40.

1.5.2. INVIO DEL VALORE DELLA LUMINOSITA' AMBIENTALE (010)

Questa operazione rientra tra le operazioni di lettura/comunicazione, può essere effettuata solo da Arduino verso la GUI.

BIT 7	BIT 6	BIT 5	BIT 4	BIT3	BIT2	BIT1	BIT0
L	SENS		VALORE [0; 31]				
0	1	0	[00000; 11111]				

La GUI deve mostrare una percentuale tra 0 e 100, mentre Arduino può inviare un valore compreso tra 0 e 31.

In Arduino, per creare una corrispondenza tra i valori [0%; 100%] e [0; 31] possiamo utilizzare la seguente formula:

NUM=31*LETTO/MAX_VALUE

Data la proporzione LETTO : MAX_VALUE = NUM : 31

Considerando che il sensore di luminosità (LDR) attribuisce alla presenza della luce un valore basso e all'assenza di luce un valore alto, per avere una scala che segue la logica
<<percentuale bassa = poca luce & percentuale alta = tanta luce>>
invertiamo il risultato facendolo diventare NUM=31-NUM.
Per ottenere un valore intero, arrotondiamo NUM per eccesso. NUM viene quindi trasformato in binario, incapsulato ed inviato alla GUI.

IN JAVA: Per riconvertire il valore inviato (da 0 a 31) in un valore percentuale [0; 100] utilizziamo la seguente formula: Math.round(VALORE*3,22) dove VALORE è l'intero 0-31 incapsulato nel byte.
Seguendo questa logica otteniamo la seguente tabella di corrispondenza (che non sarà presente nel codice in quanto implementata dalla formula).
Si può notare che i valori di percentuale sono fittizi, in quanto non corrispondono ad una reale percentuale di luminosità ma a dei valori regolati dalla tabella seguente.

VAL COMM	DOUBLE (formula)	% MOSTRATA	Note:
0	0	0	←NOTTE
1	3,22	3	
2	6,44	6	
3	9,66	10	
4	12,88	13	
5	16,1	16	
6	19,32	19	
7	22,54	23	
8	25,76	26	
9	28,98	29	
10	32,2	32	
11	35,42	35	
12	38,64	39	
13	41,86	42	
14	45,08	45	
15	48,3	48	
16	51,52	52	
17	54,74	55	
18	57,96	58	
19	61,18	61	
20	64,4	64	
21	67,62	68	
22	70,84	71	
23	74,06	74	
24	77,28	77	
25	80,5	81	
26	83,72	84	
27	86,94	87	
28	90,16	90	
29	93,38	93	
30	96,6	97	
31	99,82	100	←GIORNO

1.6. CONFERMA DEL VALORE MINIMO DI LUMINOSITA' (011)

Questa operazione rientra tra quelle di automazione ma è anche collegata al componente sensore di luminosità. Questa operazione di lettura permette ad Arduino di comunicare alla GUI l'avvenuta ricezione del messaggio di modifica del valore minimo (codice 110), che quindi farà le operazioni del caso per capire se l'operazione è andata a buon fine. Questa viene considerata anche un'operazione di automazione in quanto il concetto di valore minimo di luminosità è strettamente collegato all'automazione infatti, il microcontrollore si basa proprio su questo valore per decidere quando la luminosità è troppo scarsa a tal punto da attivare i componenti automatizzati. Il messaggio segue la tabella delle corrispondenze riportata precedentemente al punto 1.5.1.

1.7. ATTIVAZIONE DELL'AUTOMAZIONE PER TUTTI I COMPONENTI (111)

Questo messaggio molto particolare ha il compito di attivare l'automazione per tutti i componenti del sistema, siano essi LED o servo-motore, a prescindere dal loro stato. Tutti i bit contenuti nel messaggio inviato sono posti ad 1, in decimale risulta quindi il valore di 255, ovvero il massimo numero raggiungibile utilizzando 8 bit. Possiamo vedere questo messaggio come una sorta di reset, in quanto il sistema, allo stato iniziale, è in modalità automatica fino alla prima interazione dell'utente.

2. VERIFICA DELLA CONNESSIONE

Nel codice di Arduino, nella funzione loop, la prima operazione compiuta è la lettura del valore della luminosità. Subito dopo viene inviato da Arduino al programma Java il valore della luminosità, tramite il byte di cui al punto 1.5.2. Questo scambio serve anche da test di connessione tra le due parti in quanto, se la GUI non riceve alcun dato di luminosità, la connessione seriale può dirsi interrotta.

3. DESCRIZIONE DELL'INTERFACCIA GRAFICA GUI (Java-Swing)

DI FIANCO VIENE SPECIFICATO IL PROTOCOLLO UTILIZZATO

1) MOSTRARE STATO COLLEGAMENTO GUI↔INO [010]
2) MOSTRARE VALORE DELLA LUMINOSITA' CORRENTE [010]
3) N°3 PULSANTE ACCENSIONE DEI LED
    a. MOSTRA LO STATO CORRENTE (ON/OFF + AUTO ON/OFF) [000]
    b. PERMETTE DI PORLO NEGLI STATI ON/OFF/AUTO [100]
4) SELETTORE SOGLIA MINIMA DI LUMINOSITA' [110]
    a. INVIO DATI [110]
    b. RICEZIONE CONFERMA [011]
5) PULSANTE 'ATTIVA AUTOMAZIONE PER TUTTI I COMPONENTI' [111]
6) MOSTRA STATO DEL SERVO MOTORE (GRADI + AUTO ON/OFF) [001]
7) SELETTORE GRADI ROTAZIONE SERVO MOTORE [101]

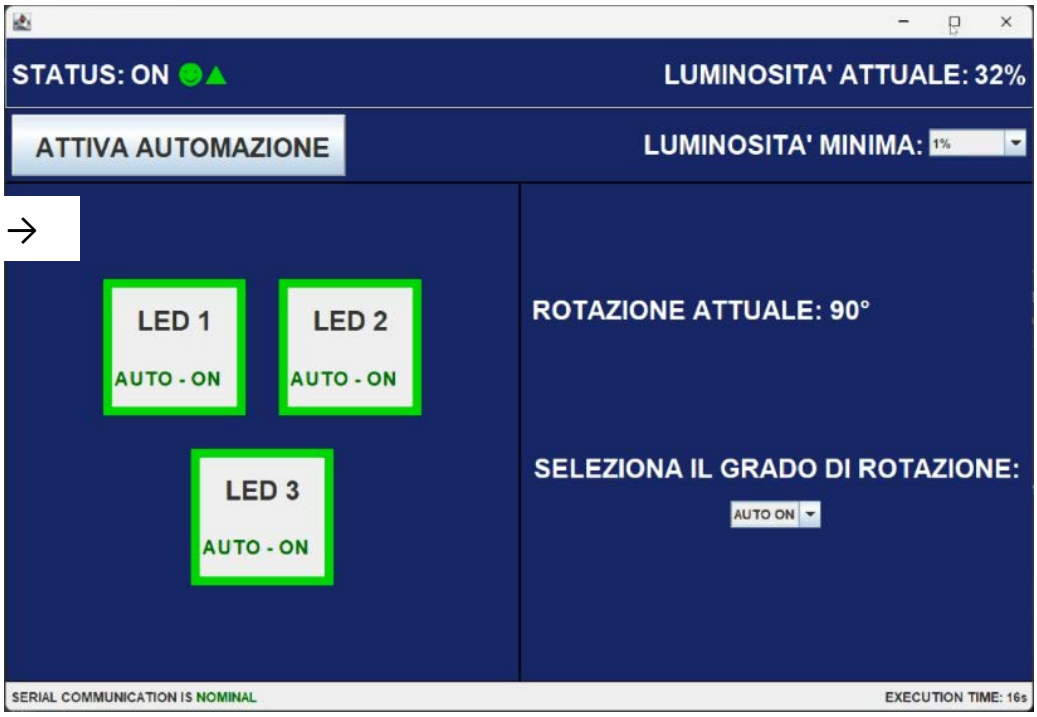
Di lato […] sono indicati i bit 7, 6 e 5 del byte inviato/ricevuto, associati al componente.

3.1. COMPONENTI SWING DELL'INTERFACCIA GRAFICA

Nel JFrame principale possiamo trovare un pannello MainPanel che a sua volta contiene 5 pannelli differenti organizzati tramite un BoxLayout. I primi 4 sono pannelli del tipo BufferedPanel, ovvero JPanel personalizzati con l'aggiunta del buffer comune (di cui parleremo in seguito) e il quinto è del tipo personalizzato FooterPanel. I pannelli sono nello specifico istanze delle classi StatusPanels, AutosPanel, LedsPanel, ServoPanel e FooterPanel.

Come è stato detto in precedenza, ad ogni pannello è associato un codice in base ai componenti che va a regolare tramite l'applicazione del protocollo. Si riportano i codici: StatusPanels=2, AutosPanel=3, LedsPanel=0, ServoPanel=1.

Di seguito è riportato lo schema dell'interfaccia grafica.



← StatusPanel

← AutosPanel

LedsPanel →

← ServoPanel

← FooterPanel

Il pannello degli stati contiene le informazioni circa il collegamento con il microcontrollore tramite seriale ed il valore della luminosità. Se la connessione seriale è presente e lo scambio di dati va a buon fine, lo stato sarà "ON" mentre se la connessione è interrotta lo stato sarà "OFF", questo cambiamento è rilevabile tramite le eccezioni lanciate dal gestore degli input. In più, grazie al Thread 'ConnectionControl' se, nonostante la connessione presente, non venissero scambiati dati per più di 15 secondi, lo status sarà cambiato in "IDLE" sino alla lettura di un messaggio da parte di Arduino contenente il valore della luminosità. Questo discorso è stato già trattato al punto 2.

Il pannello delle automazioni contiene il pulsante per attivare l'automazione per tutti i componenti ed una casella combinata per la selezione del valore minimo da adottare per la luminosità. Mediante opportuni ActionListener viene inviato il messaggio corrispondente alla scelta effettuata.

Il pannello dei LED contiene a sua volta tre pannelli del tipo SingleLedPanel disposti secondo un GridBagLayout. Ogni sotto-pannello ha un codice e rappresenta un LED fisico collegato al microcontrollore.

I SingleLedPanel contengono due pannelli, organizzati mediante un CardLayout. In più vi è un pulsante, di fatto nascosto, sotto il pannello mostrato. Una volta cliccato questo pulsante, che ricopre tutta l'area del SingleLedPanel, avviene uno scambio tra il pannello delle informazioni e quello di selezione. Nel pannello di selezione troviamo tre pulsanti "ON", "OFF" e "AUTO" che mediante opportuni ActionListener inviano il messaggio corrispondente alla scelta effettuata verso Arduino. Una volta fatta la scelta, la visualizzazione torna al pannello con le informazioni. Il bordo del SingleLedPanel varia di colore a seconda dello stato del LED correlato: se non si hanno informazioni il pannello avrà bordo nero e mostrerà "NO DATA" mentre se il LED è acceso (sia manualmente che automaticamente) il pannello avrà un contorno di color verde e, se il LED è spento (sia manualmente che automaticamente) il bordo sarà di colore rosso. Questi dati vengono ricavati volta per volta dal microcontrollore ed aggiornati (protocollo 000).

Il ServoPanel contiene una JLabel che mostra il valore corrente di rotazione del servo ed una casella combinata che permette di scegliere tra un valore di rotazione (tra quelli disponibili) da applicare al servo-motore oppure di attivare l'automazione per il suddetto componente.

Infine il FooterPanel mostra, a sinistra, lo stato della connessione seriale. Quando la connessione seriale è interrotta il sistema entra nella cosiddetta "SAFE MODE" e viene mostrato anche il tipo di errore, se l'errore è avvenuto durante la comunicazione da Java verso Arduino viene mostrata la dicitura "COM ERROR" mentre in tutti gli altri casi nominali per l'appunto viene mostrato come stato "NOMINAL".

Sulla destra è presente un cronometro che segna il tempo trascorso dall'avvio della GUI, ciò è possibile grazie al Thread "DeltaTime" che confronta il tempo attuale con quello di inizio esecuzione e ne fa una sottrazione.

3.2. BUFFER CONDIVISO

Per permettere ai vari oggetti di lavorare in modo armonioso fra di loro, si è optato per l'introduzione di un'area di memoria condivisa, detta Buffer. L'oggetto di tipo Buffer viene creato nel Main e poi passato come parametro, a cascata, in quasi tutti i pannelli ed oggetti. Per facilitare l'utilizzo del buffer nei pannelli, ad esempio, è stata creata la classe padre BufferedPanel che prende come parametro obbligatorio l'oggetto buffer ed in più il numero di protocollo assegnato al pannello (come visto in precedenza). Inoltre la classe padre definisce un metodo readBuffer() che va a leggere le informazioni ricevute dal buffer e le pone in un vettore interno.

Il buffer contiene fondamentalmente un array di booleani che contiene il messaggio inviato da Arduino a Java. Tutti i metodi del Buffer sono di tipo synchronized in quanto c'è il rischio che più consumatori vogliano accedere alle stesse risorse nello stesso momento. Fondamentale è anche il riferimento, reso così condiviso, alla porta seriale sulla quale stanno avvenendo le operazioni. Il buffer è anche alla base del BufferDataListener, che permette al sistema di diramare le informazioni in modo efficiente ai vari oggetti ascoltatori. La classe Buffer è figlia della classe BufferDataComponent che rende il buffer 'ascoltabile' da una serie di 'osservatori' memorizzati in un'apposita lista. Gli osservatori si possono 'iscrivere' a questa lista richiamando il metodo addBufferDataListener(BufferDataListener); e il buffer può notificare di un evento tutti gli ascoltatori tramite il metodo fireBufferDataChange(Object) al quale può essere passato qualsiasi tipo di oggetto. In questo modo il buffer può comunicare qualsiasi informazione, di qualsiasi tipo, ai propri ascoltatori. Il fireBufferDataChange crea un evento del tipo BufferDataEvent, il quale ha come proprietà la sorgente della 'notifica' e l'oggetto che il

Buffer vuole diramare ai suoi ascoltatori. Gli ascoltatori, che hanno implementato l'interfaccia "BufferDataListener" e che si sono aggiunti come ascoltatori al buffer, possono decidere nel proprio bufferDataListener di filtrare le notifiche utilizzando parole chiave come 'instanceOf' per capire se la natura del messaggio può essergli utile. Il BufferDataListener è una parte fondamentale del software, in quanto è l'unico strumento che collega l'interfaccia grafica alla logica di ricezione dei dati seriali.

Appena un nuovo dato arriva sul buffer, tramite il metodo setRecByte(byte), dopo la conversione del byte in array di boolean, il buffer estrapola il codice-protocollo del messaggio per capire a chi è indirizzato. Successivamente richiama il metodo fireBufferDataChange e invia il codice-protocollo del nuovo messaggio. A questo punto, tutti gli ascoltatori riceveranno il codice e, se (nel caso dei BufferedPanel) il codice corrisponderà al proprio, essi richiameranno il metodo readBuffer e leggeranno il messaggio indirizzato a sé dal buffer per poi analizzarlo e farne le opportune considerazioni. Anche il FooterPanel utilizza il BufferDataListener, in questo caso però ha un particolare interesse sulle proprietà di tipo SafeModeOutputException e SafeModeInputException lanciate come proprietà del fireBufferDataChange dai try-catch sparsi per il programma.

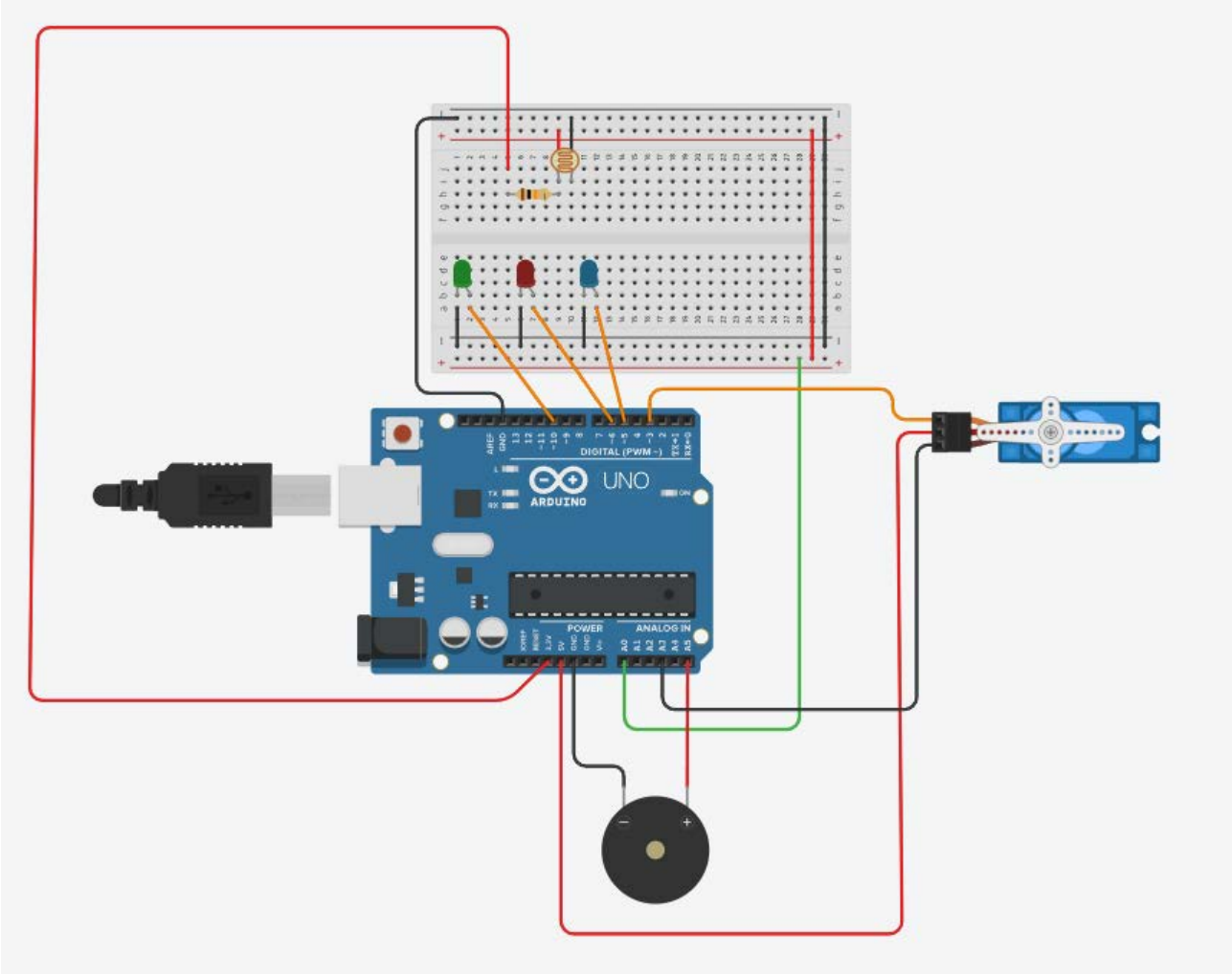
3.3. SerialCommInputManager e SendData

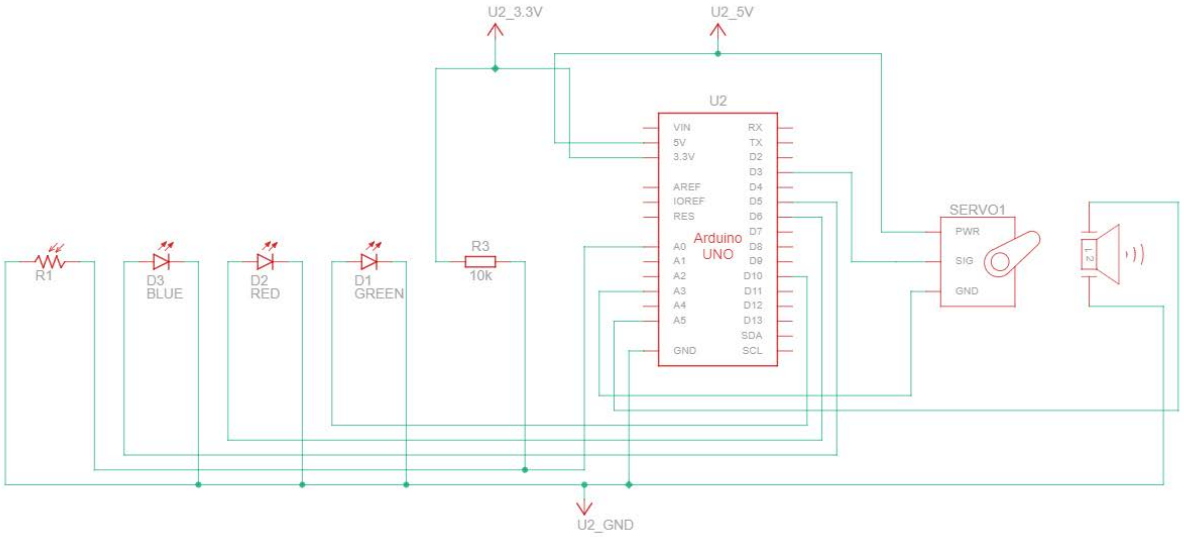
SerialCommInputManager è una classe-Thread che si occupa di leggere i dati direttamente dalla porta seriale e di inoltrarli sul buffer affinché possano raggiungere il destinatario. Nel costruttore, viene aperta la porta seriale ed il Thread avvia il proprio metodo run che scorre ciclicamente all'infinito (while(true)) e legge, istante per istante i dati dal seriale, trasferendoli poi sul buffer.

Tutte le operazioni sono svolte all'interno di vari try-catch per evitare errori, anche fatali, per il programma.
La classe SendData si occupa invece di inviare alla porta seriale (e quindi Arduino) dei messaggi. Il costruttore accetta come parametri il buffer (da cui prenderà il riferimento alla porta seriale) e il messaggio da inviare, che una volta trasformato in byte, viene inviato alla porta seriale.

La chiusura del programma avviene con l'ausilio di un WindowListener inserito nella classe GUIManager (che crea il frame) e applicato al frame. Quando viene catturato un evento di chiusura sul Frame1 (windowClosed) viene chiuso l'accesso alla porta seriale, tramite il metodo definito dal buffer (buffer.closePort()), e successivamente viene terminato il programma (System.exit(0)). Questo perché, con il metodo tradizionale di chiusura *(EXIT_ON_CLOSE),* non si è certi che la porta seriale si chiuda al terminare del processo.

4. COLLEGAMENTO DELLA SCHEDA ARDUINO UNO R3





4.1. OCCORRENTE

- Scheda compatibile con Arduino Uno R3
- Breadboard
- Fotoresistore (LDR)
- Resistore (10KΩ)
- 3 LED
- Servo-Motore SG-90
- Buzzer attivo (opzionale)
- Cavi