Diario di lavoro

Luogo	Canobbio
Data	19.02.2020

Lavori svolti

Oggi ho scritto il codice relativo ad arduino per la gestione della frequenza, lo ho testato semplicemente con il metodo **main()** della classe **SerialThread**. Il quale semplicemente setta una frequenza per il generatore, lo accende e dopo 5000 millisecondi lo spegne.

```
public static void main(String[] args) throws Exception {
    SerialThread t = new SerialThread();
    t.start();

    SerialCommand s = SerialCommand.FREQUENCE;
    s.setMessage("800".getBytes());
    t.addCommand(s);

    s = SerialCommand.GENERATOR_ON;
    t.addCommand(s);

    Thread.sleep(5000);

    s = SerialCommand.GENERATOR_OFF;
    t.addCommand(s);

    Thread.sleep(5000);
    t.interrupt();
}
```

Per avere un sistema di controllo di quello che avviene, ho deciso di aggiungere le seguenti linee al metodo **writeCommandReadResponse()**:

```
System.out.println((char)serialCommunication.getSequence());
System.out.println(new String(serialCommunication.getMessage()));
System.out.println();
```

Le quali servono per stampare la risposta del messaggio.
Dopo di che ho iniziato ad integrare il codice del primo semestre con quello sviluppato nelle settimane precedenti.

Per prima cosa ho copiato le seguenti classi nel package **ch.giuliobosco.freqline.acc**:

- SerialCommand
- SerialCommunication
- SerialEchoCommand
- SerialInputCommand
- SerialNullCommand
- SerialResponse
- SerialThread

Dopo di che ho creato la classe **MicThread** la quale serve per spegnere il generatore quando viene acceso tramite microfono.
La quale ha il metodo **run()** seguente:

```
/**
 * Wait until timer then turn off generator.
 */
@Override
public void run() {
    this.timer = System.currentTimeMillis() + timer;

    try {
        while (System.currentTimeMillis() > this.timer && !interrupted()) {
            Thread.sleep(900);
        }
    } catch (InterruptedException ignored) {

    }

    this.serialThread.addCommand(SerialCommand.GENERATOR_OFF);
}
```

La quale viene integrata nella classe **SerialMicCommand** (tramite **AccGenerator**) che implementa **SerialInputCommand**, per cui ha il seguente metodo **buildResponse()**.

```
/**
 * Build command response and update generator status.
 *
 * @return Command response.
 */
@Override
public SerialResponse buildResponse() {
    SerialResponse response = SerialResponse.OK;
    response.setMessage(this.getMessage());

    try {
        JdbcConnector connector = new JapiConnector();
        connector.openConnection();

        boolean status = GeneratorQuery.getGeneratorStatus(connector.getConnection(),
AccGenerator.KEY_C);

        if (status) {
            AccGenerator.turnGeneratorOff(connector.getConnection(),
AccGenerator.KEY_C, this.serialThread);
        } else {
            long timer = GeneratorQuery.getMicTimer(connector.getConnection(),
AccGenerator.KEY_C);
            AccGenerator.turnGeneratorOn(connector.getConnection(), AccGenerator.KEY_C,
timer, this.serialThread);
        }
    } catch (IOException | ClassNotFoundException | SQLException ignored) {

    }

    return response;
}
```

Un implementazione simile è stata fatta per la classe **SerialRemoteCommand**, mentre **AccGenerator** è stata modificata, in quanto non esegue più le richieste HTTP ma aggiunge dei comandi alla serial thread.

Infine ho creato la classe **FreqlineServer** che avvia il server e la **SerialThread** aggiungendo tutte le Servlet.

Problemi riscontrati e soluzioni adottate
Il generatore non emette correttamente tutte le frequenze, ma probabilmente è un problema di contatti.

Punto della situazione rispetto alla pianificazione
Sono in linea con la pianificazione. Attivita 12.

Programma di massima per la prossima giornata di lavoro
Aggiornare la documentazione e testare nell'insieme il progetto. (WEB + Generatore)