Relazione Progetto #3

Smart Dam

Studenti: Carletti William, Colotti Manuel Enrique, Lisotta Marco.

Introduzione

Il sistema Smart Dam è costituito da cinque sottosistemi coordinati tra loro per gestire lo stato di una diga in funzione del livello dell'acqua rilevato da un apposito sensore.
Le cinque componenti del sistema sono le seguenti:

1. **Remote Hydrometer:** Sensore che si occupa di rilevare il livello dell'acqua di un fiume e di comunicarlo agli altri sottosistemi.

2. **Dam Service:** Fondamentale per la gestione del sistema, si occupa della propagazione delle informazioni verso diverse componenti.

3. **Dam Controller:** Apparato che aziona il meccanismo di apertura e chiusura della diga in relazione alle informazioni ricevute da **Dam Service,** si occupa inoltre di inviare e ricevere dati da/a **Dam Mobile.**

4. **Dam Mobile:** Applicazione mobile che permette ad un operatore di visualizzare diverse informazioni sulla diga ed eventualmente di modificarne lo stato.

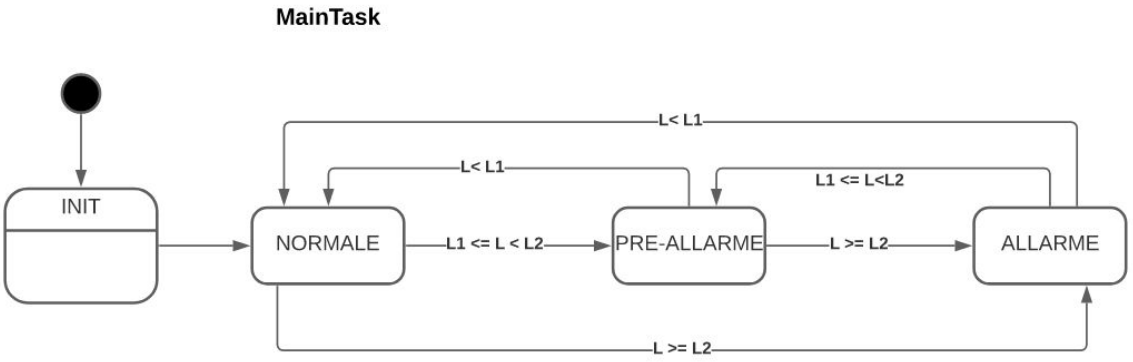
5. **Dam Dashboard:** Interfaccia Web che permette ad un utente di visualizzare dati real-time sullo stato della diga e dati storici sul livello dell'acqua.

Design del Sistema

Remote Hydrometer

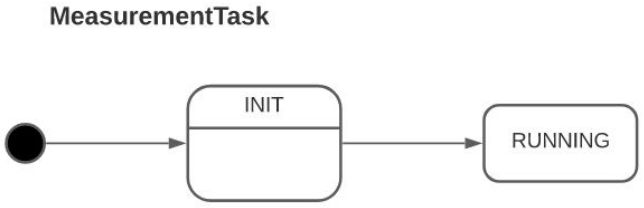
Abbiamo deciso di implementare l'idrometro come una macchina a stati finiti sincrona composta da vari **Task** che svolgono compiti diversi. I task individuati dalla suddivisione sono i seguenti:

- **MainTask:** determina lo stato corrente del sistema e attiva o disattiva gli altri task quando è necessario.

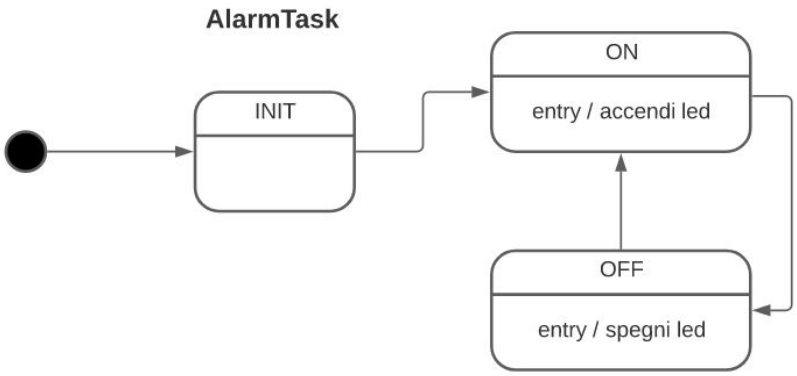


- **CommTask:** recupera l'ultimo livello dell'acqua rilevato e delega la sua trasmissione tramite **MQTT** a **MessageSender.** Questo task prevede inoltre la possibilità di modificare il proprio periodo base per poter inviare dati più velocemente in determinati stati.

- **MeasurementTask:** utilizza il sonar per rilevare il livello dell'acqua e lo comunica attraverso **WaterMeasurement** ad altri Task.



- **AlarmTask:** gestisce il blinking di un led nel caso in cui lo stato corrente sia **Pre-Alarm.**



Implementazione dello Scheduler

Per azionare i vari task in maniera sincrona abbiamo implementato uno **scheduler** che ogni **500ms**, in seguito al tick di un timer *(libreria Ticker di esp)*, esegue i diversi task uno dopo l'altro.
La scelta del periodo deriva principalmente da due fattori: in primo luogo dal fatto che la rilevazione e trasmissione del livello dell'acqua vengono effettuate ogni 5 o 10 secondi, perciò non è necessario che lo scheduler abbia un periodo troppo ridotto; il secondo fattore riguarda invece il led che in questo modo potrà lampeggiare con un periodo ragionevole di mezzo secondo.

Dam Service

Per realizzare la parte **backend** del sistema scritta in Java,
abbiamo impiegato il pattern architetturale **MVC** e ci siamo serviti del tool-kit **Vert.x** che permette un rapido deploy di diversi servizi necessari alla comunicazione tra i nostri sottosistemi.

Controller

Suddivisione in Verticle

In **Vert.x** un **Verticle** rappresenta un attore del sistema *(modello ad attori)* che in seguito alla ricezione di un messaggio può effettuare decisioni locali o anche inviare il messaggio ad altri attori. L'entry point dell'applicazione risulta essere il **MainController** il cui scopo è quello di istanziare i diversi **Verticle** ed effettuarne il deploy.
Nella parte **Controller** della nostra applicazione abbiamo individuato tre diversi tipi di **Verticle**:

1. MQTTVerticle

All'avvio effettua la subscribe su un topic *"water_level"* di un **broker Mosquitto**, questa operazione gli permetterà di ricevere i livelli dell'acqua trasmessi dal **Remote Hydrometer**. Alla ricezione di un nuovo messaggio, questo verrà passato alla parte **Model** del **Dam Service**.

2. SerialVerticle

Questo **Verticle** permette al **Dam Service** di comunicare con il **Dam Controller**, I due apparati infatti sfruttano la seriale per scambiarsi informazioni sullo stato attuale della diga *(stato e apertura)*.

- **I messaggi** spediti dal **Dam Controller** vengono prima ricevuti grazie a **SerialCommChannel,** poi inviati al **SerialVerticle;** quest'ultimo si occuperà di interpretarli e di aggiornare le relative informazioni nel **Model** del sottosistema.

- **I messaggi** inviati verso il **Dam Controller** invece vengono prima ricavati dal **Model** per poi essere analogamente immessi sulla seriale tramite **SerialCommChannel.**

3. HTTPVerticle

HTTPVerticle permette di eseguire un **Web Server** il cui scopo sarà quello di rendere disponibile alla **Dam Dashboard** tutte le informazioni di cui necessita in formato **JSON**. La creazione di quest'ultimo avviene ricavando stato e apertura della diga dal **Model,** oltre alle ultime venti rilevazioni del livello dell'acqua ottenute da un **DBMS MySQL**.

Database

In aggiunta ai **Verticle** sopracitati abbiamo deciso di implementare anche una utility class che permette al sistema di interfacciarsi con un **DBMS MySQL**. Questo risulta estremamente utile per poter salvare sul **Database** i nuovi valori rappresentanti il livello dell'acqua recuperati dal **Model.**

Model

Il **Model** è la componente del sistema che si occupa di mantenere e computare determinate informazioni relative alla diga. Nello specifico il **Model** attraverso il livello dell'acqua fornitogli da **MQTTVerticle** è in grado di ricavare lo stato attuale del sistema ed un nuovo valore per l'apertura della diga. Una volta calcolate tali informazioni, il livello dell'acqua viene inserito come record nel **DB,** mentre stato e apertura vengono resi disponibili agli altri **Verticle**.

Mosquitto Broker

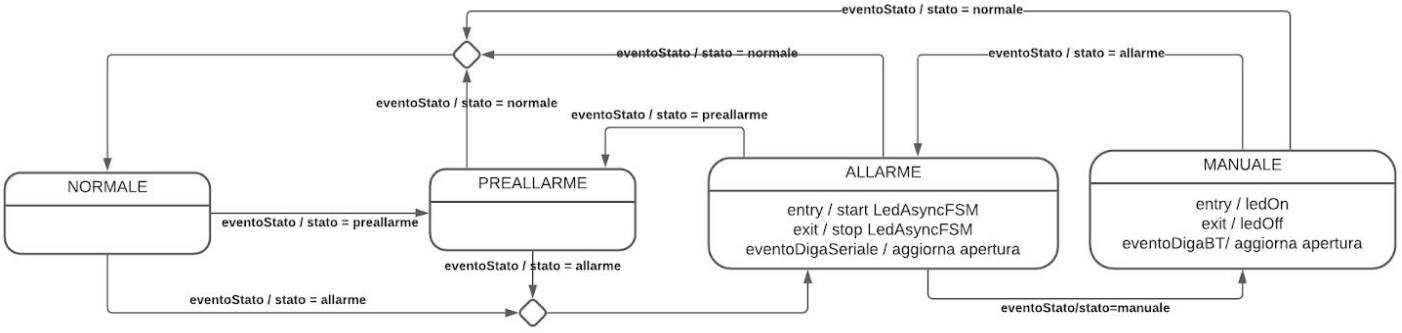
Per poter usare il protocollo applicativo **MQTT** è stato necessario usare un **broker Mosquitto** che si occupasse di mantenere in uno specifico topic ("*water_level*") i dati sul livello dell'acqua passatigli da **Dam Hydrometer** e che fosse in grado di renderli disponibili a uno o più degli altri sottosistemi.

Dam Controller

Il sistema **Dam Controller** è stato realizzato come sistema asincrono, in quanto il suo compito principale risiede nel reagire a determinati eventi generati principalmente dalle componenti **Dam Service** e **Dam Mobile**.
Perciò, sono state create due macchine a stati asincrone denominate **MainAsyncFSM** e **LedAsyncFSM.**

MainAsyncFSM

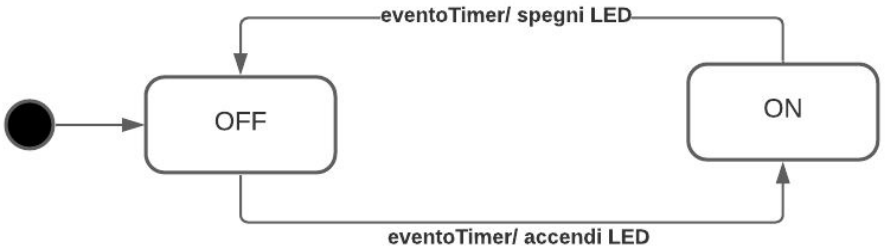
Questa **FSM** rappresenta il cuore di **Dam Controller**: essa, infatti, si occupa della gestione dei messaggi ricevuti sia via **Bluetooth** da parte del sottosistema mobile, che via **seriale** *(da parte del DamService)*. In base a questi messaggi, la macchina controlla l'apertura della diga *(rappresentata come il movimento di un servomotore)*, lo stato in cui la **FSM** si trova *(gestendo di conseguenza attivazione/disattivazione di un led)* e la propagazione di informazioni da **Mobile** a **Service** e viceversa.
Tutti i messaggi ricevuti da questa macchina sono rappresentati come **eventi**: per questo, il canale **Bluetooth** ed il canale **Seriale** sono stati modellati come **Event Source** il cui observer è la **MainAsyncFSM** stessa.



*Rappresentazione stati della **MainAsyncFSM***

LedAsyncFsm

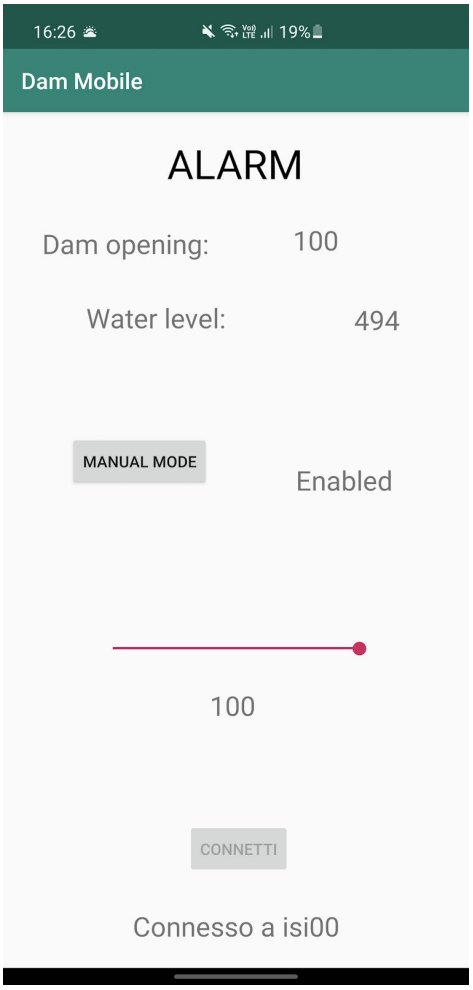
Questa semplice **macchina a stati asincrona** si occupa del blinking di un led. Essa viene attivata/disattivata dinamicamente da **MainAsyncFSM**, a seconda dello stato in cui si trova. Per ovviare al fatto che il cambiamento dello stato di questa macchina avviene periodicamente, è stato usato come fonte di eventi un **Timer**: ad ogni tick del timer, il sistema passerà allo stato opposto.



*Rappresentazione stati della **LedAsyncFSM***

Dam Mobile

Il sistema **Dam Mobile** è stato realizzato come un semplice applicativo Android: una volta connesso al ricevitore **Bluetooth** di **Dam Controller**, esso permette all'operatore di visualizzare lo stato generale della diga, l'ultima misurazione del livello dell'acqua e l'apertura della diga.
Nel caso in cui il sistema si trovi in allarme, verrà mostrato all'utente un pulsante per attivare/disattivare la modalità **manuale**, modalità che consente la modifica dell'apertura della diga mediante uno slider.
I dati relativi al livello dell'acqua, al contrario di stato e apertura, non vengono recuperati mediante **Bluetooth**: vengono infatti ricavati utilizzando il protocollo **MQTT**, previa connessione al broker e sottoscrizione all'apposito topic "*water_level*".
Abbiamo scelto questo approccio per evitare che il **Dam Controller** debba gestire un informazione a lui non funzionale, collegando quindi l'applicazione direttamente alla sorgente dell'informazione richiesta.

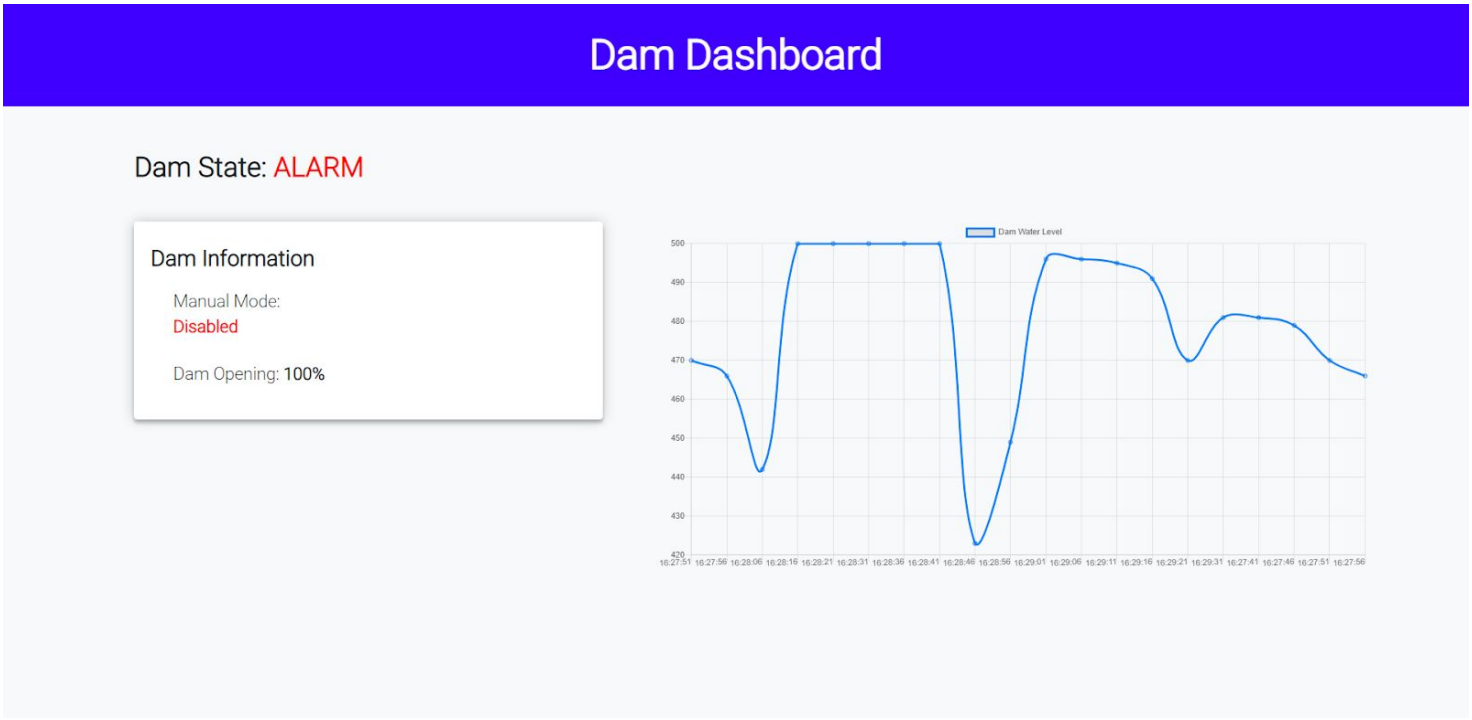


*Schermata principale dell'applicazione **Dam Mobile***

Dam Dashboard

La dashboard viene utilizzata per la visualizzazione di tutte le informazioni raccolte sullo stato attuale della Diga.
Per realizzarla abbiamo utilizzato una pagina **HTML** aggiornata dinamicamente grazie all'impiego di **JavaScript**.
Quest'ultimo si occupa di effettuare richieste **HTTP GET** con cadenza periodica *(5 secondi)* al **Dam Service**. Quest'ultime vengono prese in carico dal Verticle HTTP *(come già spiegato precedentemente)* restituendo un file di tipo **JSON** contenente tutte le informazioni da visualizzare in relazione allo stato attuale del sistema.

Per la visualizzazione degli ultimi venti valori del livello dell'acqua abbiamo utilizzato una **libreria di javaScript (chart.js)** che permette la creazione dinamica di un grafico sulla base di un dataset.



Schema elettrico

Di seguito lo schema elettrico di Dam Controller e Remote Hydrometer.