Progetto Smartcities
Sistema di monitoraggio ambientale

UPO

2018

Indice

1 Introduzione

Obbiettivo del progetto è quello di esplorare di la possibilità di costruire una rete capillare di stazioni di monitoraggio ambientale che sia a basso costo e facilmente dipiegabile sul territorio.
La soluzione cui si è pervenuti si basa su un'architettura ispirata al paradigma dell'*Internet of Things* (IoT nel seguito) ed incentrata su un *message broker* localizzato in *cloud*. Al message broker si interconnettono sia stazioni di raccolta dati semplici ed a basso costo basate su Arduino e Raspberry PI che servizi cloud capaci di aggregare i dati raccolti dalle stazioni e di rilevare l'insorgenza di situazioni critiche, quali la presenza di gas tossici e di focolai di incendio.

Come caso di studio, si sono sviluppati due tipi di stazioni, uno ad installazione fissa specializzata per il rilevamento di fumi e di fiamme, l'altro mobile, da installarsi su autoveicoli, specializzato per il rilevamento di polveri sottil e gas nocivi alla respirazione.
Le stazioni sono affiancate da un insieme minimale (per ora) di servizi di elaborazione dati che individuano le aree geografiche in cui si verifica una situazione di emergenza ed offrono all'utente la possibilità di analizzare lo stato ambientale aggiornato in tempo reale.
L'accesso ai servizi è implementato da un server REST che funge da intermediario tra il browser dell'utente e i servizi cloud suddetti.
Nel seguito, descriveremo brevemente l'architettura del sistema, i servizi implementati e le API REST per l'accesso da parte dell'interfaccia utente.

2 Architettura del Sistema

Il sistema di raccolta e processamento dati è stato progettato seguendo la classica architettura dei sistemi IoT allo stato dell'arte. La struttura centrale dell'architettura è il *Message Broker*, ovvero un gestore di mailbox localizzato in rete pubblica, tramite il quale agenti localizzati in reti pubbliche o private (purchè connesse ad Internet) possono scambiarsi messaggi.
La maggior parte de Message Broker si basano sul protocollo di comunicazione MQTT, originariamente proposto da IBM per la raccolta di dati di telemetria [**?**] ed ora adottato come standard di fatto nel settore IoT. Esistono a disposizione in rete diversi Message Brokers, alcuni inclusi in piattaforme come Azure [**?**], altri direttamente accessibili come ad esempio RabbitMQ [**?**] o Mosquitto [**?**].

Mosquitto ha il pregio di essere facilmente installabile su Linux, anche su host di basso profilo come Raspberry Pi [**?**] e Beaglebone [**?**]. Inoltre Mosquitto ha la caratteristica di scalare facilmente a grandi numeri di agenti connessi sfruttando il cosiddetto meccanismo del *bridging*. Usando questo meccanismo è possibile configurare reti gerarchiche di brokers che coprono vaste reti geografiche instradando i messaggi secondo necessità.

L'architettura IoT che ne deriva è del tipo di quella descritta in Figura 1.

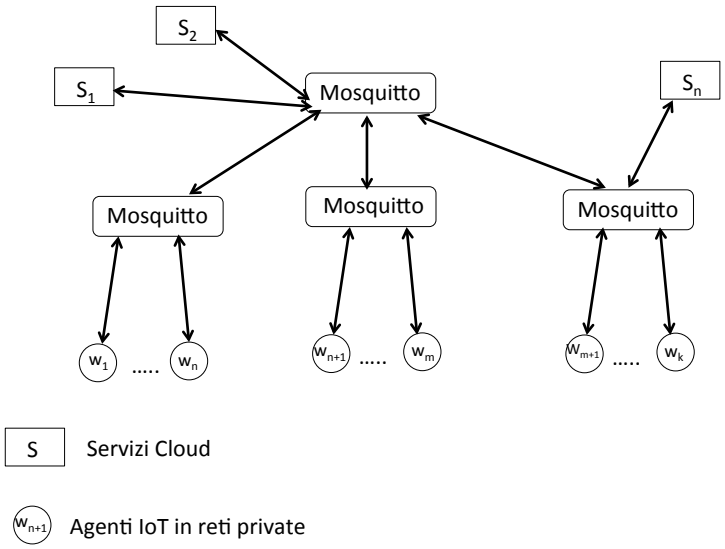

Figure 1: Architettura IoT basata su Message Broker

La comunicazione tra agenti e message brokers MQTT, avviene seguendo il paradigma *publish subscribe* [**?**]. Implicitamente si definisce un albero tassonomico di soggetti che verranno usati per identificare il tipo di contenuto di un messaggio. Un messaggio è sempre associato a un tipo corrispondente ad una foglia dell'albero.

Un agenti che deve comunicare dati, deve sempre scegliere un tipo, corrispondente a una foglia "T" dell'albero, da assegnare al messaggio che verrà inviato al broker con una operazione di *publish*.

Un agent che vuole ricevere messaggi si registra *subscribe* sul broker specificando un path sull'albero dei tipi e rimane in attesa. Se specifica una foglia "T" dell'albero riceverà solo messaggi di tipo "T". Se, invece, specifica un path che termina ad un nodo non-foglia "t", riceverà messaggi corrispondenti a tutti i tipi che condividono il path fino al nodo "t".

L'architettura specifica del nostro progetto è descritta in Figura

In particolare esistono due tipi fondamentali di agenti: le stazioni di raccolta dati che sono distribuite sul territorio e i servizi cloud che sono allocati in su server di rete e quindi possono utilizzare grandi quantità di calcolo.

2.1 Architettura implementata

Mosquitto + rete di Mosquitto
MongoDB
OpenStreetMap
Stazioni di Raccolta
Tecnologie WEB

3 Stazioni di raccolta

3.1 L'architettura

L'architettura della rete utilizzata per implementare il nostro progetto prevede diverse componenti hardware e software, ognuna con uno scopo diverso. Alla base troviamo il broker; server fondamentale per la trasmissione di dati tra stazioni di rilevazione e software "back-end".

Nello specifico, le stazioni sono formate come segue:

- **Rasperry Pi** (nella versione 3B)
- **Arduino Nano**
- **Sensori:** sensore di fumo, sensore di fiamma, sensore di qualità dell'aria e sensore di concentrazione di particolato.

Partendo dal particolare, troviamo le schede Arduino: queste, collegate ai sensori, si occupano di: prelevare i dati ambientali, formattarli e prepararli al passaggio successivo.
Ogni Arduino implementa delle tecniche che permettono di evitare errori e gestire letture errate (vengono ignorati ad esempio dati non validi). Con questo approccio è possibile gestire errori sul nascere ed evitare problemi in futuro. È stato scelto Arduino perché supportato da creatori e sviluppatori, ma soprattutto perché permette la lettura di dati analogici in modo semplice.
Per assicurare una maggiore affidabilità, alla scheda vengono affidati esclusivamente compiti relativi alla lettura dei dati.

È da notare che, sebbene Arduino gestisca la maggior parte dei sensori, per le stazioni mobili viene utilizzato un modulo GPS che si collega direttamente tramite USB alla scheda Raspberry. Sarebbe stato possibile collegarlo all'Arduino per mantenere più rigida la struttura gerarchica dei dispositivi, tuttavia una buona generalizzazione del software e l'elasticità di implementazione ci hanno permesso di utilizzare tale approccio.

Ora, tramite una connessione USB seriale, i dati vengono inviati alla scheda Raspberry.

La Raspberry si occupa di prelevare dati, di verificarne la consistenza, di elaborarli se necessario e quindi di formattarli perché seguano uno standard definito in fase di progettazione.
La scheda, essendo dotata di una buona capacità di calcolo, viene sfruttata per qualunque tipo di elaborazione preliminare. Essendo molto potente, non ci sono problemi di latenza tra l'invio di un dato e quello successivo.
La scelta di una Raspberry è stata quasi immediata, data la sua diffusione, il suo supporto hardware e software ed i consumi ridotti.

3.2 Il software

La stesura del software ha seguito un approccio di agile development. Si è quindi passati attraverso diverse fasi di prototipazione fino ad arrivare alla struttura finale.
Il focus principale durante lo sviluppo è stato quello di mantenere il software il più modulare e generalizzato possibile. In questo modo ci è stato possibile ridurre al minimo le modifiche necessarie per renderlo utilizzabile su entrambi i tipi di stazione. Viene da sé che sarà possibile implementare altri tipi di stazione senza dover riprogettare l'intera struttura software.

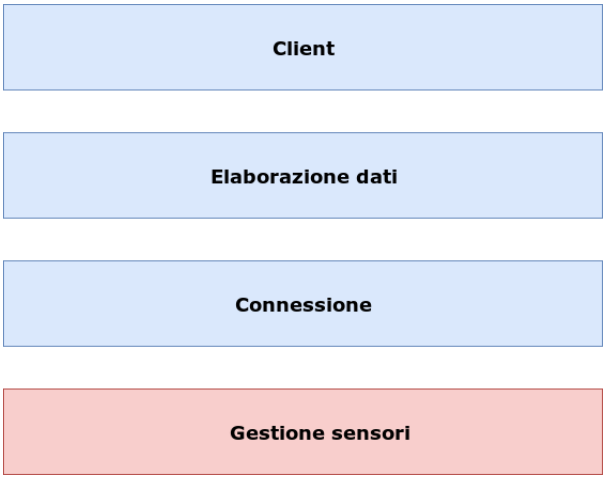

Figure 2: Struttura generale del software. L'implementazione differisce in base al tipo di stazione

Nello specifico, il software è suddiviso come in Figura 2.

Il **livello client** si occupa di gestire il "comportamento" della stazione. Fondamentalmente, riceve i dati dai livelli sottostanti e li invia al broker, gestendo nel mentre le connessioni all'hardware, alla rete e assicurando la corretta esecuzione di ogni parte del codice.
La parte di **elaborazione dati**, si occupa di formattare i dati e di inserirli correttamente nel formato JSON. In questo modo, si assicura la standardizzazione

dei dati facilitando il lavoro del server di analisi.
Il **livello di connessione** gestisce la comunicazione seriale con i dispositivi collegati alla Raspberry. In particolare, questo livello è in grado di riconoscere univocamente i dispositivi collegati e di leggere dati in arrivo sul canale seriale.
La **sensoristica** viene gestita da Arduino. Questo, attraverso codice molto essenziale, legge i dati dai sensori e li invia tramite la connessione seriale.

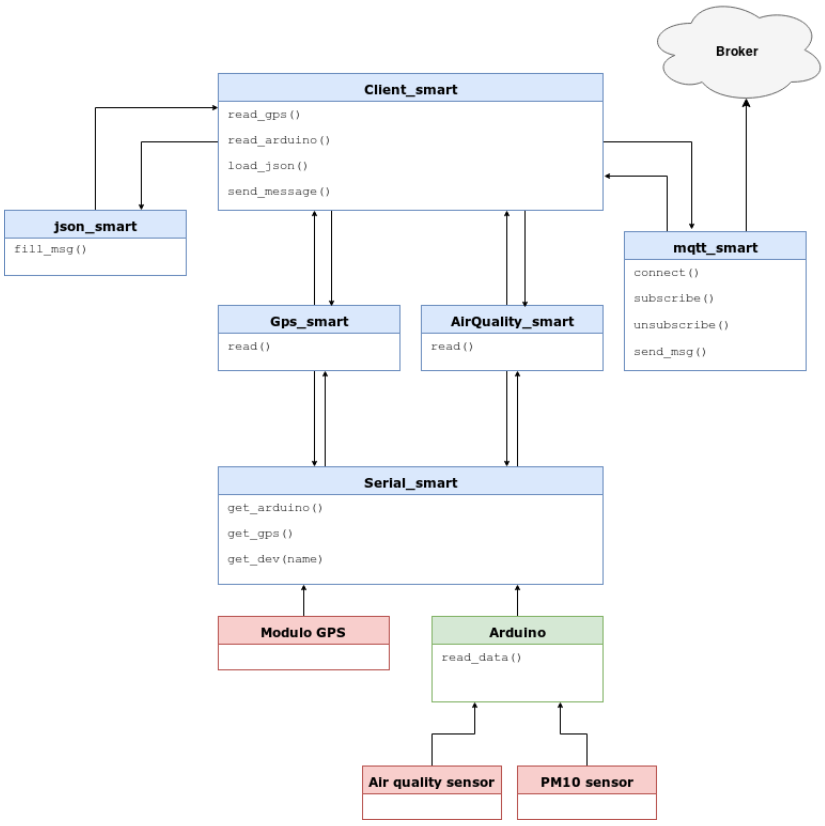

Figure 3: Struttura software per stazioni mobili.

La struttura software per le stazioni mobile è rappresentata in Figura 3 e riceve dati da tre sensori differenti:

- **Modulo GPS:** invia informazioni sulla posizione in formato GPGGA attraverso il canale seriale.
- **Sensore di qualità dell'aria:** sensore collegato all'Arduino che fornisce un indice sulla qualità dell'aria basato sulle concentrazioni di vari gas nocivi presenti nell'atmosfera.
- **Sensore di particolato:** sensore collegato all'Arduino che fornisce la concentrazione di PM10 nell'aria.

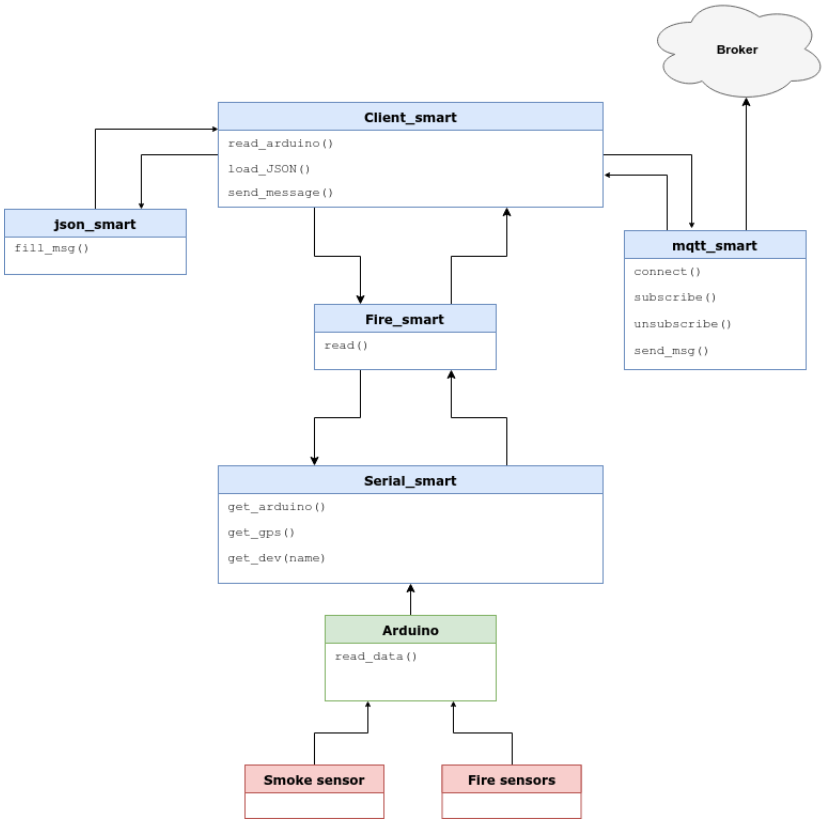

Figure 4: Struttura software per stazioni fisse.

La struttura software per le stazioni mobile è rappresentata in Figura 4 e riceve dati da due sensori:

- **Sensore di fiamma:** sensore in grado rilevare la presenza di fiamme nelle vicinanze.
- **Sensore di fumo:** sensore in grado di rilevare la concentrazione di fumo nell'atmosfera.

3.3 Varianti possibili

La struttura hardware descritta precedentemente per le stazioni di raccolta dati è solamente una delle possibili soluzioni architetturali. Infatti, è possibile apportare delle modifiche hardware per adattarsi a situazioni particolari, senza dover rivoluzionare il lato software dell'architettura.

4 Funzionalità cloud

Rappresentazione concettuale dello scenario Aggregazione dati Completamento dati nelle aree non monitorate Previsione evoluzione futura

5 API

Descrizione dettagliata delle API

Le API (application programming interface) rivestono un ruolo importante all'interno del progetto. Esse permettono di interfacciarsi alle funzionalità real time e di storico dati dell'applicativo. Data l'architettura scelta, è stato opportuno operare dividendo in 2 categorie le API:

- **API Real Time:** queste API forniscono l'accesso tramite web socket ai dati che vengono pubblicati dal message broker. Quest'approccio permette di avere in tempo reale i dati delle rilevazione selezionate.
- **API per lo storico:** queste API permettono di accedere ai dati storicizzati per mese, anno ecc.

5.1 API real time

Real Time è la parola chiave di questo applicativo. Fornire in tempo reale zone ad alto inquinamento ed il propagarsi di un incendio è fondamentale. La tecnologia che permette questo sono i websocket.
Un websocket permette di instaurare un canale di comunicazione tra i client ed il server centrale. Il funzionamento è molto semplice: Per prima cosa il meccanismo si basa sul server, il quale deve essere sempre attivo è gestire N client in parallelo. Un client, per instaurare il canale di comunicazione con il server, deve effettuare una connessione TCP.
Per realizzare questo meccanismo client-server con i websocket, è stata utilizzata la libreria javascript `Socket.io`.
Questa libreria permette di realizzare una connessione full-duplex basata su eventi.

- **Lato Server:** la libreria `Socket.io` è stata integrata con il server Node. Lato server implementiamo le funzionalità di connessione, invio messaggio e disconnessione. La connessione avviene tramite l'evento 'connection' e viene codificata quanto segue:

```
io.on('connection',(client) => {
// Azioni per il client
});
```

`io` è l'istanza di `socket.io`, in attesa di una connessione da un altro socket. All'interno della funzione, tramite il parametro client, è possibile identificare il client ed implementare l'invio di messaggi dal server ad esso.

```
client.emit('message',{msg: JSON.parse(message)});
```

Tramite la funzione emit, possiamo inviare messaggi al client. Una volta fatto il parsing del messaggio, viene inviato sul canale.

```
io.on('disconnect',function(){
    console.log('Client disconnesso');
});
```

Tramite l'evento disconnect, sappiamo quando un client si disconnette.

- **Lato Client:** Per quanto riguarda il client, la libreria da caricare è `socket.io-client`.

```
const socket = openSocket('hostname:porta');
```

Tramite la funzione `openSocket` possiamo aprire una connessione verso l'indirizzo e porta inserito come argomento (nel nostro caso il server).

```
socket.on('message', (msg) => {
//Azioni per il msg
});
```

All'interno di questa funzione, tramite l'evento message possiamo ricevere i messaggi inviati dal server e gestirli. msg conterrà il messaggio.

Il meccanismo descritto sopra fornisce le API per le funzionalità real time. Chiunque voglia ricevere i dati in tempo reale, ha solo bisogno di aprire la connessione verso il server tramite un websocket.

5.2 API per lo storico dati

È importante avere lo storico di tutte le rilevazioni passate per profilare al meglio una zona. Per fare ciò, è necessario seguire un approccio RESTful. Dato che l'applicazione deve fornire solo la richiesta di un set di dati, verranno creati degli URL endpoint per permettere questa azione. I dati sul database saranno una quantità molto elevata e bisogna fornire la possibilità di accedervi con criteri temporali predefiniti.

URL endpoint:

- `hostname/aria/year`: tutti i dati dell'anno corrente.
- `hostname/aria/month`: tutti i dati del mese.
- `hostname/aria/day`: tutti i dati del giorno corrente

Tramite una GET possono essere richiesti diversi set di dati.

6 Interfaccia utente

L'interfaccia utente è stata implementata utilizzando le tecnologie web più utilizzate al momento nel campo dello sviluppo frontend, quindi linguaggio JavaScript e il framework React. L'applicazione si compone di quattro parti principali:

- `Monitoraggio Real-Time stazioni Aria`
- `Monitoraggio Real-Time stazioni Fuoco`
- `Vista dettagliata livelli di inquinamento su mappa`
- `Analisi dati storici Aria`
- `Analisi dati storici Fuoco`

6.1 Monitoraggio Real-Time stazioni Aria

In questa sezione vengono mostrate le segnalazioni provenienti dalle stazioni in funzione tramite un marker disegnato sulla mappa. [IMMAGINE ALESSANDRIA CON MARKER DI PIU' STAZIONI] E' possibile cliccare su di un marker in modo da visualizzare il nome della stazione e i livelli di PM10 e QoA rilevati in quel punto specifico; Inoltre è possibile decidere di seguire una stazione in particolare cliccando sul link "Segui" nel popup che compare quando si clicca su un marker. [IMMAGINE MARKER CON POPUP] Una volta stabilita la stazione da seguire, verranno mostrate solo le segnalazioni di quella stazione sulla mappa e il centro della mappa verrà di volta in volta spostato sull'ultima segnalazione ricevuta, in modo da avere ben chiaro il percorso compiuto dalla stazione stessa. [IMMAGINE STAZIONE SEGUITA] Tramite il pannello di comandi disponibile è possibile smettere di seguire una stazione (cliccando sull'icona a forma di croce) oppure fermare il monitoraggio, in modo da non ricevere più le nuove segnalazioni.

6.2 Monitoraggio Real-Time stazioni Fuoco

6.3 Livello inquinamento su mappa

In questa sezione viene mostrato all'utente il livello di QoA in ogni zona della città, mediante dei rettangoli di diverso colore, scelto in base al livello raggiunto dalla zona. [IMMAGINE MAPPA CON COLORI] Questo risultato viene raggiunto tramite l'uso di una matrice che ricopre l'intera città, i quali singoli elementi corrispondono ad un'area di circa 170 metri quadri; Consultando questa matrice e il suo contenuto (ogni cella contiene un valore di QoA ottenuto tramite interpolazione e il colore dell'intervallo in cui ricade il valore) è possibile mostrare all'utente la situazione della QoA in modo semplice ed intuitivo.

6.4 Analisi dati storici Aria

In questa sezione viene mostrata la mappa della città nello stesso modo della sezione 6.3, ma utilizzando i dati relativi al momento storico scelto dall'utente, il quale può specificare una data da dopo la quale considerare i dati contenuti nel database NoSQL MongoDB. [IMMAGINE SEZIONE MAPPA STORICO] Tramite il pannello di comandi è possibile specificare la data d'interesse e selezionare se considerare tutte le rilevazioni del periodo selezionato oppure solo quelle anomale/pericolose. [IMMAGINE GRAFICO STORICO] Inoltre tramite l'apposito tasto presente nel panello dei comandi è possibile visualizzare un grafico che mostra l'andamento del PM10 e della QoA nel periodo d'interesse.

6.5 Analisi dati storici Fuoco

7 Valutazione del sistema

Modello teorico dell'incendio Modello della raccolta inquinamento con automezzi Dati reali: raccolta con automobile Valutazione ottenuta

8 Conclusioni e sviluppi futuri

9 Bibliografia

10 Appendici