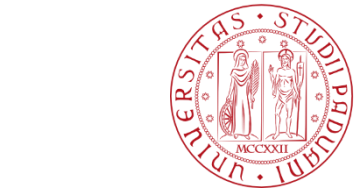

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO MATEMATICA

TerraSense – Documento di Specifiche

Mirco Borella

Mat. 2075530

Progetto di Programmazione ad Oggetti
LT in Informatica
Università degli Studi di Padova

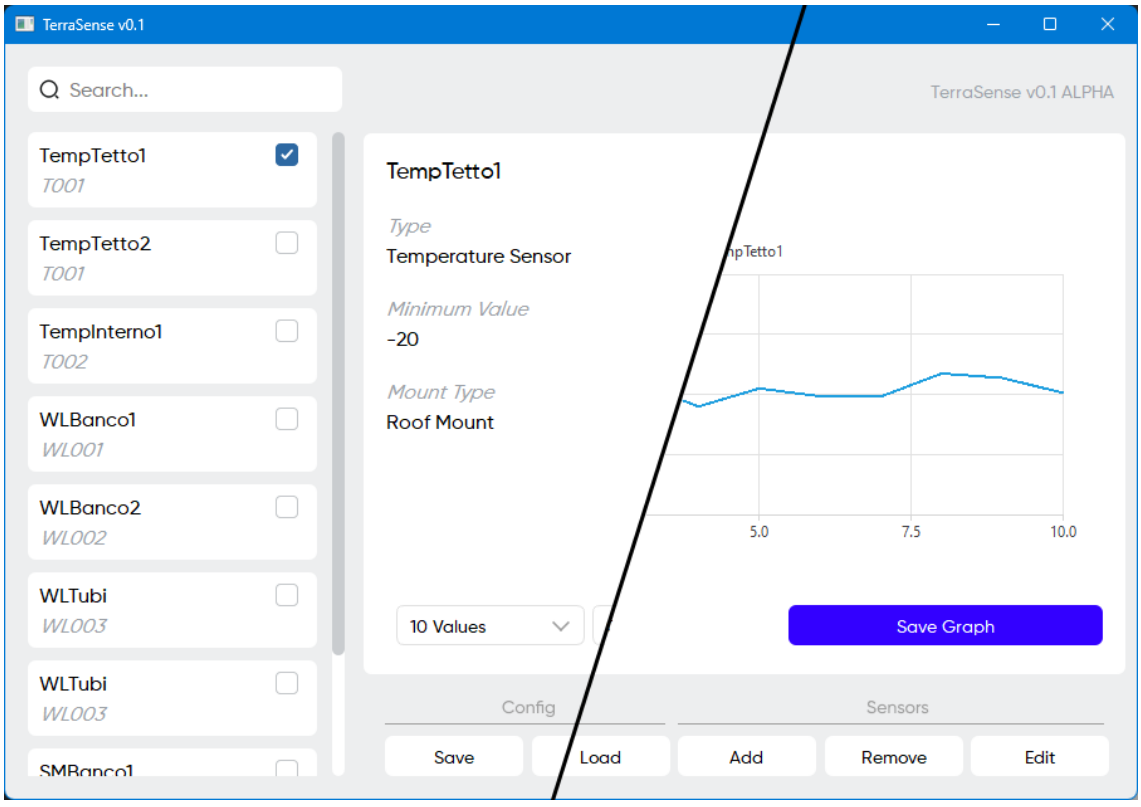

Figura 1: Anteprima interfaccia grafica della piattaforma TerraSense

Indice

1 Introduzione

TerraSense è una piattaforma per la gestione e la simulazione di sensori all'interno di una serra. Il software permette di aggiungere, modificare, cancellare e cercare sensori idealmente installati fisicamente nello stabile, e di simularne l'andamento dei parametri attraverso diverse distribuzioni di dati.

Ogni sensore è dotato di un nome, un gruppo, un valore minimo e un valore massimo (che compongono il range di utilizzo). Ogni tipologia di sensore ha due campi propri.
Nello specifico, sono state implementate le seguenti tipologie di sensore, contenenti i seguenti campi dati propri:

- *Sensore di temperatura*, avente come campi propri la tipologia di montaggio (montaggio a muro / sul tetto / da tavolo) e il tipo di alimentazione (corrente elettrica / solare / batteria).
- *Sensore di livello dell'acqua,* avente come campi propri il livello di protezione (IP68 / IPX5 / IP57) e la tipologia di misura (ultrasonica / fisica / galleggiante).
- *Sensore di umidità del terreno,* avente come campi propri il materiale del sensore (acciaio / rame / ottone) e la profondità di installazione (10cm / 20cm / 45cm).

È possibile avere più sensori con lo stesso nome e stesso gruppo. Ogni parametro potrà poi essere modificato selezionando il sensore da modificare attraverso il sistema di checkbox e premendo il tasto *Edit*.

L'uso della piattaforma inizia dalla lista di sensori inseriti sulla sinistra. Ogni sensore dispone di una checkbox per la selezione. È possibile effettuare operazioni diverse in base al numero di sensori selezionati, ad esempio selezionando un solo sensore, verranno mostrati i relativi parametri e il menu per la generazione dei dati. Selezionando più sensori e premendo il tasto *Remove*, verrà cancellata tutta la lista selezionata.

A sensori caricati, è possibile cercare uno specifico nome o gruppo di un sensore attraverso la barra di ricerca. La ricerca è case-sensitive e permette l'utilizzo di RegEx e

relativi operatori, come ad esempio: “.” , “@” oppure “[A-Za-Z]”. È possibile cancellare e aggiungere sensori quando è in corso una ricerca. Il sensore aggiunto verrà mostrato se appartenente alla ricerca.

Attraverso i tasti *Save* e *Load*, è possibile salvare l’intera configurazione di sensori in un file JSON per poi poterla ricaricare in un secondo momento. Tutte le modifiche non precedentemente salvate verranno perse alla chiusura dell’applicativo.

Selezionato un sensore, è possibile avviarne la simulazione scegliendo tra 3 quantitativi di dati da simulare (10, 20 o 30 valori) e tra 3 tipologie di distribuzioni dati (casuale, gaussiana e logaritmica). Generato un grafico, è possibile esportarlo in formato PNG premendo il relativo tasto. Verranno esportati in un’immagine l’andamento e il nome del sensore.
È stato scelto di sviluppare la piattaforma in lingua inglese.
Molteplici risorse sono state impegnate nell’interfaccia grafica, soggetta di un particolare studio e sviluppo per essere utilizzata attraverso schermi touch-screen dotati di tastiera.

2 Descrizione del design pattern e del modello logico.

La piattaforma è stata sviluppata utilizzando il design pattern MVC (Model-View-Controller). La classe *Controller* si occupa del routing di informazioni tra modello e vista. Sono state utilizzate diverse tipologie di segnali e slot per permettere la comunicazione tra controller e vista, i quali sfruttavano la possibilità di inviare valori, puntatori ad oggetti o vettori di dati condivisi tramite std::make:shared<> per far ricevere al controller l’input dell’utente.
La classe *MainWindow* svolge la funzione di *view* e si occupa di mostrare informazioni e ricevere l’input dall’utente. Essa è stata sviluppata per essere un componente separato e facilmente intercambiabile. La classe *SensorLogic* svolge la funzione di modello e gestisce alcuni dati e la logica.
Secondo il design pattern MVC, il modello può essere in grado di aggiornare parti della vista accedendoci direttamente. Questo è stato sviluppato permettendo alla vista di ricevere puntatori a sensori, salvandoli all’interno dei relativi widget grafici contenitori.
L’intera sezione dedicata alla ricerca di sensori è stata sviluppata nella vista, in quanto caratteristica offerta proprio dalla vista implementata. Altri tipi di UI e/o viste possono non avere questa funzionalità.

SensorLogic è responsabile del mantenere la lista dei sensori attualmente inseriti nell’applicativo. Il controller mantiene solo la lista dei sensori selezionati attraverso la vista e si occupa di gestire tutte le varianti come l’aggiornamento dei menu della vista o il sensore da inviare al modello per eseguirne operazioni su di esso come la modifica dei parametri.

Il modello logico della piattaforma è sviluppato in due parti. La prima, comprende le classi che descrivono le tipologie di sensori.

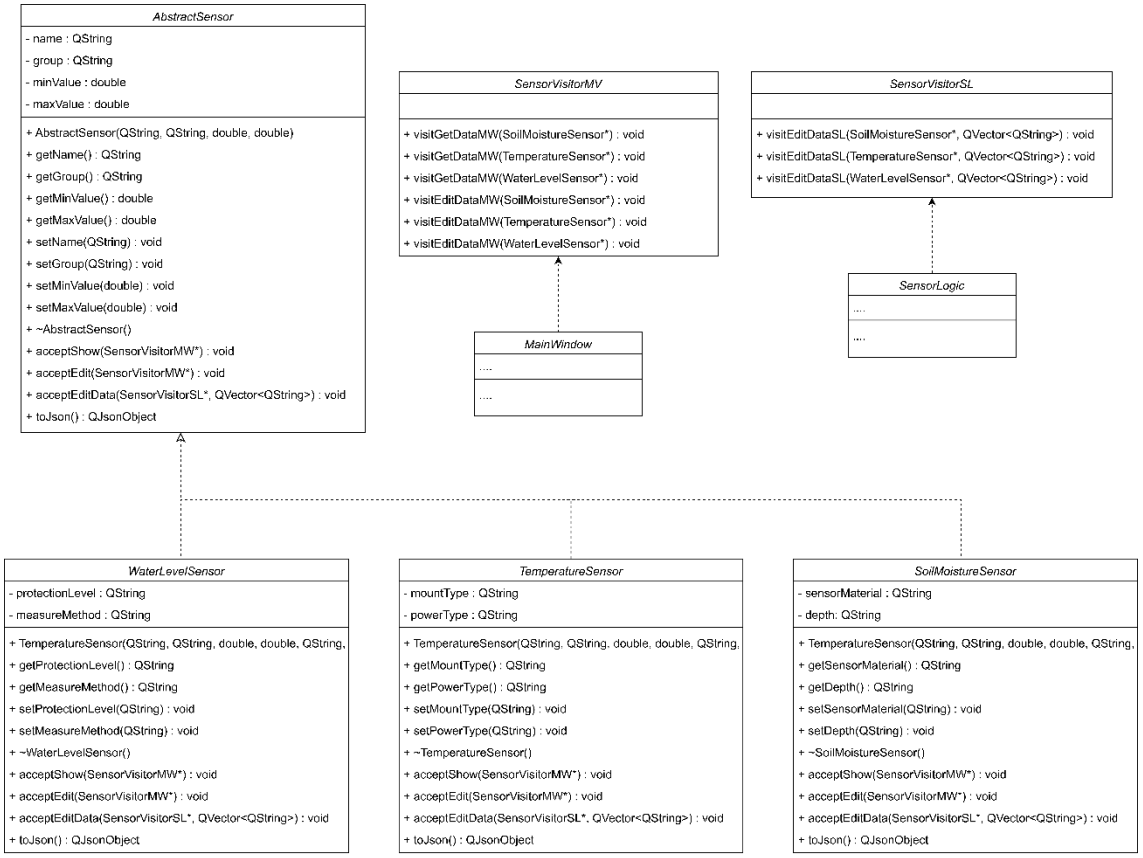

Figura 2: Diagramma delle classi del modello logico

La classe *AbstractSensor* è la classe base astratta che contiene tutte i parametri comuni tra le varie tipologie di sensori. Essi sono il nome, il gruppo ed il range di utilizzo, composto da un valore minimo ed un valore massimo. La classe, oltre ai relativi metodi getter e setter per le variabili d'istanza, contiene il metodo virtuale puro pubblico *toJson()*, necessario per produrre in seguito la persistenza dei dati.

Le classi *TemperatureSensor*, *WaterLevelSensor* e *SoilMoistureSensor* sono classi figlie di *AbstractSensor* e, oltre ad implementare il metodo *toJson()*, dispongono anche dei metodi getter e setter per gli attributi propri (descritti al punto *1 - Introduzione*) di ogni tipologia di sensore.

La seconda parte del modello riguarda il salvataggio in formato JSON dell'array di sensori salvato in *SensorLogic.* Ulteriori informazioni al punto *4 – Persistenza dei dati*.

3 Descrizione del polimorfismo non banale

L'impiego principale del polimorfismo riguarda due design pattern *Visitor* nella gerarchia *AbstractSensor.* Il primo visitor pattern denominato *SensorVisitorMW* viene utilizzato nella vista per:

- mostrare la corretta scheda di informazioni. Le varie tipologie di sensori possiedono parametri propri che necessitano di alcuni accorgimenti per riuscire a visualizzarli correttamente.

- mostrare la corretta scheda di modifica. Selezionando un sensore tramite checkbox dalla sezione a sinistra e premendo sul tasto *Edit* è possibile accedere alla scheda dedicata alla modifica. Anche questa scheda varia in base alla tipologia di sensore selezionato.

Il secondo visitor pattern denominato *SensorVisitorSL* viene utilizzato nel modello in concomitanza con il primo visitor per permettere la modifica effettiva dei parametri.

Le classi che quindi implementano un design pattern Visitor sono *MainWindow* e *SensorLogic.* In generale i visitor sono stati utilizzati per riconoscere la tipologia di sensore.

4 Persistenza dei dati

Come già citato in precedenza, per la persistenza dei dati si è optato per utilizzare un unico file in formato JSON.
Attraverso l'implementazione del metodo *toJson()*, le tipologie di sensori offrono un modo per ritornare un *QJsonObject*, che rappresenta una array chiave-valore di ogni parametro del sensore. Inoltre, aggiunge un'ulteriore chiave "sensorType" specifica che tipologia di sensore si sta salvando. Segue implementazione della coppia key-value con chiave "sensorType": `json["sensorType"] = "TipologiaDiSensore"`

Il metodo di *SensorLogic saveToJSON()*, dopo aver fatto scegliere il nome dell'output del file attraverso una finestra di dialogo di sistema all'utente, scorre l'array dei sensori chiamando il metodo *toJson()* e aggiunge ogni *QJsonObject* ad un *QJsonArray* che in fine verrà salvato.

Il metodo di *SensorLogic loadToJSON()*, ricarica da un file una configurazione dei sensori precedentemente salvata, andando a sovrascrivere l'attuale configurazione di sensori caricata nell'applicativo.

Tutte le modifiche non salvate verranno perse alla chiusura. Il tasto *Save* dell'interfaccia grafica non è disponibile se non ci sono sensori caricati nella piattaforma.

Si riporta una configurazione di esempio *Example.json* nella radice del progetto contente vari sensori (con varie combinazioni anche di nomi uguali e gruppi uguali) che consente un test immediato dell'applicativo.

5 Elenco e descrizione delle funzionalità implementate

Si riporta un elenco delle principali funzionalità funzionali implementate:

- Creazione, rimozione, modifica di tre tipologie di sensori
- Ricerca di sensori case-sensitive tramite RegEx in tempo reale (pressione del tasto *Enter* non necessaria per effettuare la ricerca)

- Salvataggio della configurazione dei sensori su file in formato JSON
- Caricamento della configurazione dei sensori da file in formato JSON
- Salvataggio del grafico della simulazione dei sensori in formato PNG
- Tre tipologie di distribuzione dati per le simulazioni (Random, Logaritmica e Gaussiana)
- Scorciatoie da tastiera per le principali azioni (funzionanti solo se il relativo tasto del menù è attivo):
    - CTRL+A per aggiungere un sensore *(Add)*
    - CTRL+R per rimuovere i sensori selezionati *(Remove)*
    - CTRL+E per modificare il sensore selezionato *(Edit*)
    - CTRL+S per salvare la configurazione dei sensori *(Save)*
    - CTRL+L per caricare la configurazione dei sensori *(Load)*

Si riporta un elenco delle principali funzionalità estetiche implementate:

- Interfaccia grafica singola (non vengono utilizzate ulteriori finestre, finestre di dialogo di sistema a parte) *(figura 1)*
- Design adattato per sistemi touch-based
- Ogni componente è stato stilato tramite QSS per garantire continuità alla piattaforma (ad eccezione di finestre di dialogo di sistema per il salvataggio in JSON e PNG, il caricamento da JSON e possibili alert box di errore) *(figura 1)*
- Hover sui pulsanti
- Dimensione dell'interfaccia dinamica. È richiesta una risoluzione minima della finestra dell'applicativo di 900x600. È possibile ingrandire l'interfaccia fino a dimensione indefinita.
- Sistema di selezione di sensori grafico tramite checkbox *(figura 3)*
- Menu delle azioni dinamico in base al numero di sensori selezionati *(figura 4)*
- Interfaccia progettata in concomitanza allo sviluppo attraverso l'utilizzo di software dedicati alla UI/UX come Figma *(figura 5)*

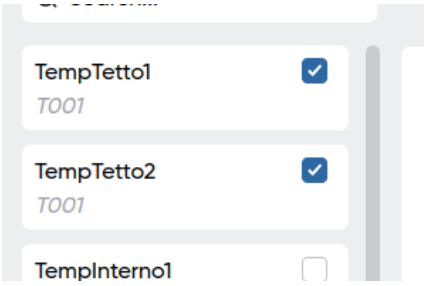

Figura 3: Selezione tramite checkbox

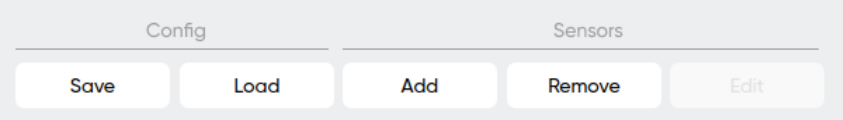

Figura 4: Menu delle azioni dinamico in base al numero di sensori selezionati (2 sensori selezionati)

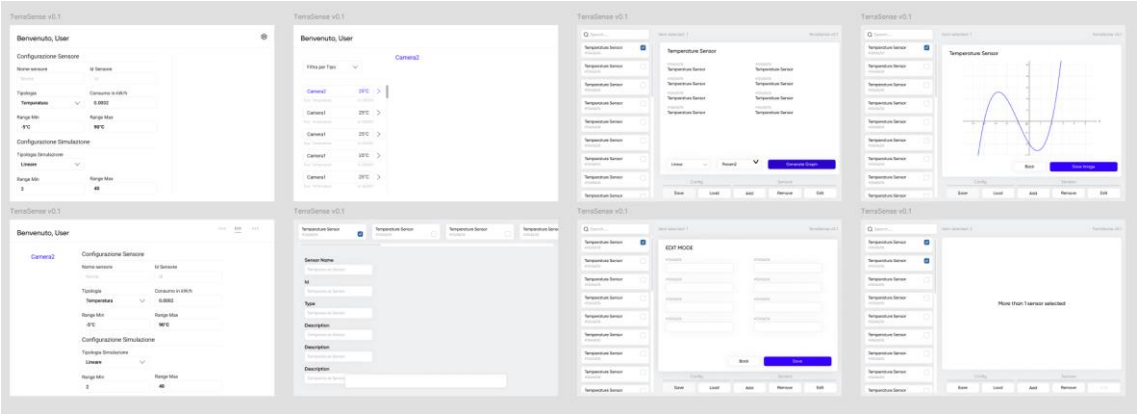

Figura 5: Workspace di Figma durante il design dell'interfaccia

6 Rendicontazione ore

Attività	*Ore Previste*	*Ore Effettive*
Studio e progettazione grafica	5	9
Studio del framework Qt	8	14
Sviluppo del codice del modello	10	11
Sviluppo del codice della GUI	18	23
Test e debug	5	7
Stesura della relazione	5	5
Totale:	51	69

Il monte ore stimato è stato superato di circa 18 ore.

La differenza maggiore tra le ore previste e quelle effettive la ritroviamo per l'attività "Studio del framework Qt", la quale comprende delle ore di puri esperimenti con il framework. Non avendo mai sviluppato avvalendomi del framework Qt, mi è servita una "test platform" sulla quale fare pratica nell'implementare widget grafici e i collegamenti tra di essi. Le ore apparentemente perse in questa attività si sono rivelate ben spese al momento dell'implementazione della GUI, che ha impegnato il maggior numero di ore dedicate al progetto, ma che non ha avuto rallentamenti degni di nota.

Anche la fase di "Studio e progettazione grafica" ha impiegato più del previsto.
Nelle 9 ore, sono state progettate 8 bozze grafiche per passione e per studio del linguaggio visuale, completando poi quella più promettente *(esempio in figura 5)*.

Sviluppando in parallelo codice del modello e codice della GUI, riuscivo ad avere un riscontro quasi immediato delle funzionalità implementate.

È stato riscontrato un solo problema degno di essere espresso nella fase “Test e debug”.
L’applicativo è stato sviluppato in un sistema operativo diverso da quello fornito nella macchina virtuale utilizzata per la correzione. Testando in fase finale il software nella VM, è stato riscontrato che sulla macchina virtuale sembrerebbe ci sia un bug grafico dove la finestra dell’applicativo a volte rimane nera, o contiene alcune sezioni nere. Questo bug, seppur (nei miei test) molto raro, è facilmente aggirabile riavviando l’applicativo.
Dopo ricerche e analisi, credo che il motivo di tale problema sia da attribuire al gestore delle macchine virtuali Virtualbox che non supporta nativamente sui miei sistemi l’accelerazione hardware grafico.

Inoltre, nella macchina virtuale è installata una versione di Qt Framework diversa da quella usata durante lo sviluppo. Alcuni metodi che nei miei sistemi non davano *warning* li davano invece nella vm. Si riporta l’esempio del file SensorLogic.cpp, dove a riga 138 il metodo *.midRef()* è stato sostituito da *.mid()* per l’antecedente motivo.

7 Ulteriori sviluppi

Una funzionalità importante da implementare, ma tralasciata per non superare ancora di più il monte ore, per rendere l’applicativo “production-ready” è la gestione degli errori nelle schede di inserimento e modifica dei sensori.
Attualmente i campi “Minimum Value” e “Maximum Value” non hanno un *error handling* adeguato, ed è possibile inserire valori minimi maggiori dei valori massimi o addirittura stringhe di testo nei due campi. Da questa necessità, sarebbe consono sviluppare una gestione degli errori appropriata e la relativa interfaccia grafica che segnala in caso di errore i campi da correggere.

Una seconda funzionalità da implementare sarebbe il selettore di più tipologie di grafico e un modo per generare grafici che confrontano più sensori dello stesso tipo.

L’ultima funzionalità pensata e non implementata riguarda il poter rimostrare una simulazione passata e salvarla durante l’export nel file JSON. In questo modo quando si ricarica una configurazione dal file JSON, si ricaricano anche tutte le simulazioni dei sensori effettuate in precedenza.