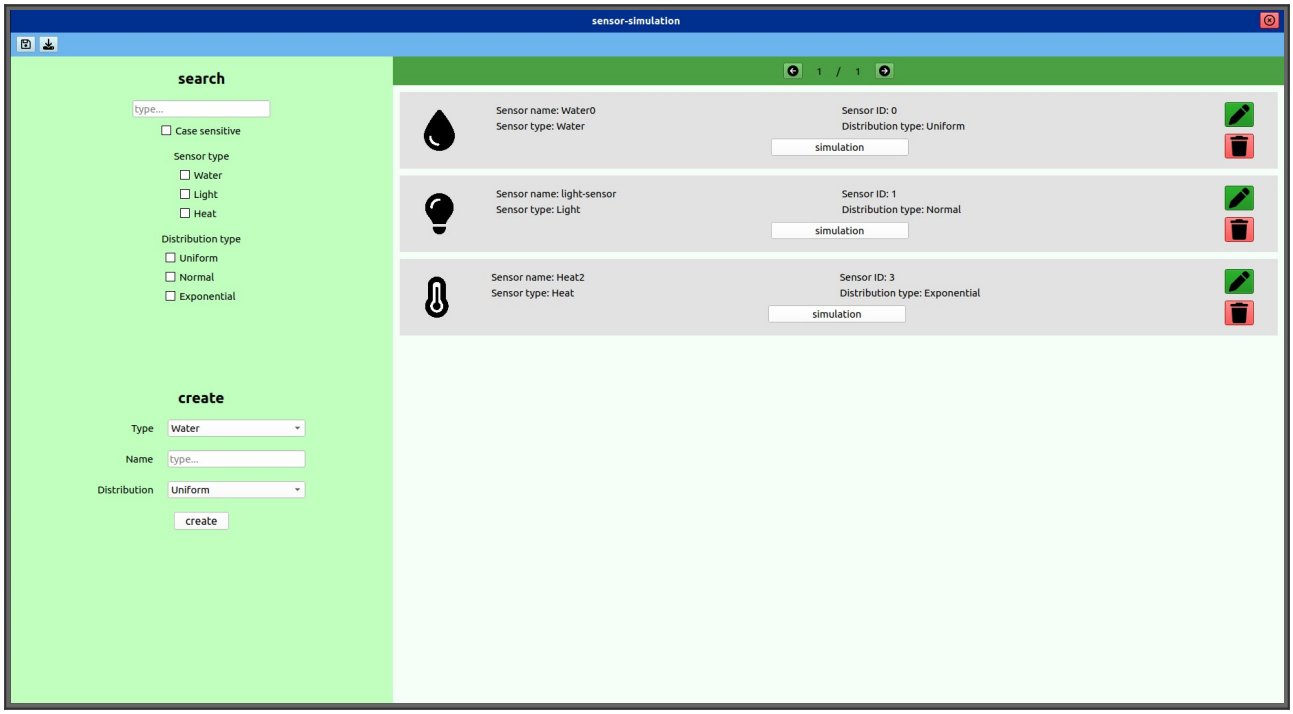
oggetto Relazione progetto Programmazione a Oggetti

gruppo Precoma Andrea, mat. 2068227

titolo Greenhouse Monitoring



Introduzione

Greenhouse Monitoring è un software che simula la gestione di sensori adibiti a monitorare una serra. Permette di creare, modificare, cancellare e visualizzare i sensori, salvare il sistema sensoristico su un file e caricarne uno già esistente. Ogni sensore è capace di generare una simulazione di dati raccolti e renderli visibili tramite un grafico in base alla distribuzione scelta.

Vi sono tre tipi di sensori: uno che monitora l'acqua erogata per innaffiare le piante, uno che monitora l'intensità di luce che si percepisce all'interno della serra, infine uno che monitora la temperatura. Ciascun sensore inoltre espone delle informazioni aggiuntive: rispettivamente il primo aggiorna l'utente circa quanta acqua è stata riciclata dalle piante che non hanno assorbito l'intero dosaggio nell'ora precedente, il secondo segnala la necessità di azionare delle luci per mantenere un minimo livello di illuminazione giornaliera, il terzo allerta l'utente se sono state rilevate temperature oltre i limiti consentiti. I dati che i sensori generano una volta avviata una simulazione possono seguire tre tipi di distribuzione: la prima è uniforme, la seconda è normale e la terza è esponenziale.

La simulazione restituisce i dati monitorati ogni ora a partire dalla mezzanotte. Si può modificare la distribuzione con la quale generare i dati e il numero di ore trascorse dalla mezzanotte oltre che ai parametri della distribuzione scelta.

Descrizione del modello

Per una più facile comprensione si consiglia la visione del grafico UML della parte logica allegato (logic.dia).

Ciascun sensore è descritto dalla classe astratta *Sensor* che viene concretizzata nella classe derivata del tipo del sensore. La classe *Sensor* contiene gli attributi comuni a tutti i sensori, i più importanti sono un ID univoco inizializzato con una variabile locale statica, la distribuzione adottata e il vettore di dati generati per la simulazione; da notare che l'attributo *typeName* ha il solo scopo di facilitare la rappresentazione del tipo del sensore sulla GUI. Sono definiti inoltre i comportamenti principali quali i getter e setter, l'avvio di una simulazione e la sua interruzione, il controllo dei parametri aggiornabili dall'utente e la gestione dei design pattern *Observer* e *Visitor*. L'osservatore è dotato di un metodo che notifica l'avvio della simulazione e di uno che aggiorna la GUI dopo una modifica dei parametri. Infine cinque metodi *accept* sono definiti per permettere le seguenti funzionalità secondo il design pattern *Visitor*: la ricerca con utilizzo di filtri, l'ottenimento e l'assegnazione dei parametri specifici di ciascun sensore, la visualizzazione della simulazione e la scrittura del sensore in formato JSON.

I tre tipi di sensori si distinguono oltre che per i parametri da monitorare anche per le informazioni fornite e le funzionalità legate al proprio tipo specifico: *WaterSensor* contiene i metodi che simulano la quantità di acqua erogata ad ogni ora e la percentuale riciclata dalle piante che ne hanno assorbito solo in parte precedentemente; *LightSensor* possiede i metodi che in base al livello di luminosità minimo della serra accende l'impianto interno; *HeatSensor* definisce dei limiti massimi e minimi di temperatura oltre i quali allerta il personale.

Le distribuzioni sono descritte da una classe astratta *Distribution* che vengono concretizzate nelle classi derivate del tipo di distribuzione. La classe *Distribution* contiene gli attributi comuni a tutte le distribuzioni quali il numero di campioni generati, ii limiti minimo e massimo; l'attributo *typeName* copre le stesse funzionalità di *typeName* del sensore. Sono definiti inoltre i comportamenti principali quali la clonazione, i getter e setter, la generazione di una serie di dati, il controllo dei parametri aggiornabili dall'utente e la gestione del design pattern *Visitor*. *Infine q*uattro metodi *accept* sono definiti per permettere le seguenti funzionalità secondo il design pattern *Visitor*: la ricerca con utilizzo di filtri, l'ottenimento e l'assegnazione dei parametri specifici di ciascuna distribuzione e la scrittura della distribuzione in formato JSON.

La classe *SensorsManager* è definita come Singleton per assicurare l'istanziazione di un solo manager; possiede due mappe per memorizzare i valori di default di ciascun sensore e distribuzione, i getter necessari, implementa la logica relativa alla creazione, rimozione e ricerca dei sensori e gestisce il design pattern *Observer*.

Per il salvataggio e il caricamento dei file JSON è stata creata la classe *JsonManager* anch'essa come Singleton; contiene gli attributi fondamentali dei file quali il percorso, il nome e lo stato (modificato o salvato), i getter e setter, i metodi *save* e *load* e gestisce il design pattern *Observer*. In appoggio al metodo di salvataggio è stato implementato il design pattern *Visitor* in modo da catturare i dati di ciascun sensore e distribuzione e scriverli nei file, invece il metodo *load* usufruisce della funzione *readSensor* (dichiarata nel file jsonreader.h) che a sua volta utilizza delle funzioni per leggere i dati dai file in base all'oggetto interessato. Si è pensato di mantenere le funzioni di lettura semplicemente in un file e non in una classe dedicata perché non necessitano di attributi e non sarebbe stato ottimale dover istanziare un oggetto ad ogni lettura. Il file risulta equivalente ad una classe che possiede il metodo di lettura pubblico e le funzioni di appoggio private.

Infine la classe *ErrorManager* contiene tutte le classi utilizzate per lanciare specifici errori.

La parte grafica è strutturata dal pannello di ricerca e creazione dei sensori, il catalogo posto centralmente, la barra delle azioni coi pulsanti di salvataggio e caricamento dei file e infine la barra per le azioni riguardanti la applicazione. Il pannello di modifica e di simulazione appaiono come finestre all'interazione di un pulsante.

Per l'interazione tra le classe della GUI e quelle legate alla logica sono stati previsti diversi design pattern *Visitor* e *Observer* descritti più precisamente nel paragrafo successivo.

Vengono posti inoltre controlli sui valori dei parametri dei sensori e delle distribuzioni, sull'apertura di un file senza aver salvato quello corrente, sulla creazione di un sensore che non può essere visualizzato perché è stato applicato un filtro alla ricerca, alla chiusura dell'applicazione quando il file corrente non è stato salvato. Al fine di comunicare in modo adeguato all'utente queste segnalazioni è stata implementata una piccola gerarchia per visualizzare a schermo i messaggi di errore e di avvertimento.

Polimorfismo

L'utilizzo del polimorfismo è visibile nella logica legata ai sensori e alle distribuzioni, alla scrittura in file JSON e ai vari utilizzi dei design pattern *Visitor* e *Observer*.

Le classi *Sensor* e *Distribution* presentano un metodo virtuale *validParams* col compito di ritornare una eccezione nel caso i dati inseriti dall'utente non soddisfino i criteri che ciascuna classe (base e derivata) impone. Nella classe *Distribution* è presente inoltre la logica *clone* per la copia delle distribuzioni e il metodo *generateDistribution* virtuale puro che restituisce il vettore di dati secondo la distribuzione specifica della classe derivata; in questo modo si può accedere al sensore richiesto, quindi alla distribuzione adottata e infine richiamare il metodo che restituirà i dati secondo la distribuzione del sensore.

Sono state implementate tutte le seguenti logiche relative al design pattern *Visitor*:

- Filtro dei sensori in base al nome, tipo e distribuzione adottata (*IFilterVisitor*)
- Cattura delle informazioni relative allo specifico sensore e distribuzione per poterle inserire nel pannello delle modifiche (*ISensorInfoVisitor* e *IDistributionInfoVisitor*)
- Settaggio dei parametri relativi allo specifico sensore e distribuzione per poterli aggiornare tramite il pannello delle modifiche (*ISensorSetVisitor* e *IDistributionInfoVisitor*)
- Personalizzazione della simulazione secondo le specifiche del sensore (*ISimulationVisitor*)
- Scrittura degli attributi del sensore e della distribuzione in formato JSON (*IJsonWriterVisitor*)

Sono state implementate anche tutte le seguenti logiche relative al design pattern *Observer*:

- Apertura del pannello delle simulazioni quando un sensore ne avvia una e aggiornamento del pannello del sensore quando vengono modificati il nome e la distribuzione (*ISensorObserver*)
- Aggiornamento del catalogo alla creazione, eliminazione e ricerca di un sensore nonché all'apertura di un nuovo file (*ISensorsManagerObserver*)
- Aggiornamento del nome del file corrente visualizzato sulla upperbar al salvataggio e alla modifica dello stesso (*IJsonManagerObserver*)

Una parte di polimorfismo è stata applicata anche alle classi riguardanti i messaggi della GUI. La classe base *MessageLog* definisce gli attributi comuni a tutti i log insieme al metodo *showMessage* virtuale puro cosicché ogni classe derivata personalizzi il pannello in base alle funzionalità.

Persistenza dei dati

Per la persistenza dei dati viene utilizzato il formato JSON, un unico file per catalogo di sensori, contenente un vettore di oggetti. Ciascun oggetto rappresenta un sensore all'interno del quale vengono definiti campi semplici chiave-valore e un oggetto che rappresenta la distribuzione adottata, anch'essa con dei campi semplici. I tipi di sensori e di distribuzioni sono identificati da due attributi "type" utilizzati per la serializzazione delle sottoclassi. Un esempio è fornito dal file "sensor-simulation.json" fornito insieme al codice con alcuni esempi di sensori.

Funzionalità implementate

Le funzionalità implementate non immediatamente visibili sono, per semplicità, suddivise in due categorie: funzionali ed estetiche. Le prime comprendono:

- gestione di tre tipologie di sensori e di distribuzioni
- conversione e salvataggio in formato JSON
- ricerca tramite filtri
- assegnazione automatica di un nome di default univoco ad un sensore creato senza aver specificato il nome
- avvertenze personalizzate per ogni simulazione in base alla tipologia del sensore

Le funzionalità grafiche sono:

- upperbar con pulsante per la chiusura dell'applicazione
- scorciatoie da tastiera (CTRL+S per salvare, CTRL+O per aprire un file)
- personalizzazione dei grafici (unità di misura e titolo) e delle avvertenze a seconda della tipologia di sensore
- visualizzazione del nome del file aperto sulla upperbar con l'aggiunta del simbolo "*" nel caso il file sia stato modificato ma non salvato
- Messaggi di avvertenza in caso si inseriscano valori non idonei nel pannello delle modifiche, nel caso di apertura di un file o di chiusura dell'applicazione mentre quello corrente non è stato salvato
- utilizzo di colori e stili grafici
- utilizzo di un file .qrc con alias

Le funzionalità elencate sono intese in aggiunta a quanto richiesto dalle specifiche del progetto.

Rendicontazione ore

Attività	Ore Previste	Ore Effettive
Studio e progettazione	10	12

Sviluppo del codice del modello	10	27
Studio del framework Qt	10	13
Sviluppo del codice della GUI	10	15
Test e debug	5	4
Stesura della relazione	5	3
totale	50	**74**

Il monte ore è stato superato in quanto sono state riscontrate difficoltà nello sviluppo del codice e nella creazione della GUI. In particolare sono stati di difficile comprensione alcuni comportamenti del framework Qt poiché non apparivano errori sulla console bensì semplicemente non si visualizzava a schermo quello che si era pianificato. Inoltre è stata necessaria una restaurazione di alcune parti del progetto perché mi sono accorto più tardi che alcuni requisiti non erano soddisfatti.

Differenze rispetto alla consegna precedente

Rispetto alla precedente consegna è stata rivisitata la logica del design pattern Visitor. Sono state implementate quattro superclassi (due per i visitor costanti e due per quelli non costanti): *ISensorVisitor* e *ISensorConstVisitor* come interfacce dei sensori, *IDistributionVisitor* e *IDistributionConstVisitor* come interfacce delle distribuzioni; in questo modo le classi *Sensor* e *Distribution* contengono solo due metodi *accept* l'una. I precedenti visitor e interfacce sono stati modificati, rimossi o sostituiti nel seguente modo:

- *IFilterVisitor*: rimosso, *FilterVisitor* ora eredita direttamente *ISensorConstVisitor* e *IDistributionConstVisitor*
- *ISensorInfoVisitor* e *IDistributionInfoVisitor*: sostituiti rispettivamente da *ISensorConstVisitor* e *IDistributionConstVisitor*
- *IDensorSetVisitor* e *IDistributionSetVisitor*: sostituiti rispettivamente da *ISensorVisitor* e *IDistributionVisitor*
- *ISimulationVisitor*: sostituito da *ISensorVisitor*
- *IJsonWriterVisitor*: rimosso, *JsonWriterVisitor* ora eredita direttamente *ISensorConstVisitor* e *IDistributionConstVisitor*