

Corso di Laurea Magistrale in **Ingegneria Informatica**

**Progetto Greenhouse**

**Elaborato finale per il Progettazione e Sviluppo di Sistemi Software**

Prof. Anna Rita Fasolino

A.A. 2019/2020

**Studenti**:

Andrea Marchetta M63001044

Marco Barletta M63001018

Riccardo Corvi M63001003

Sommario

[1. Introduzione 3](#_Toc51145448)

[2. Documento di visione per progetto Greenhouse 4](#_Toc51145449)

[3. Glossario di visione 7](#_Toc51145450)

[4. Use Case model document 8](#_Toc51145451)

[4.1. Ricerca una coltivazione 10](#_Toc51145452)

[4.2. Modifica i parametri ambientali target 14](#_Toc51145453)

[5. Specifiche supplementari 20](#_Toc51145454)

[6. Fase di elaborazione: diagrammi di alto livello 22](#_Toc51145455)

[6.1. Domain model 22](#_Toc51145456)

[6.2. Grasp model 22](#_Toc51145457)

[6.3. Sequence Diagram di analisi – Ricerca coltivazioni 23](#_Toc51145458)

[6.4. Sequence Diagram di analisi – Modifica parametri ambientali target 26](#_Toc51145459)

[6.5. Activity diagram di analisi – Controlla parametri ambientali 27](#_Toc51145460)

[7. Architettura del sistema 30](#_Toc51145461)

[7.1. Overview dell’architettura 30](#_Toc51145462)

[7.2. Componente MainFrame 32](#_Toc51145463)

[7.3. Componente ControlUnit 34](#_Toc51145464)

[7.3.1. Comunicazione 34](#_Toc51145465)

[7.3. Sequence Diagram di dettaglio – Ricerca coltivazioni 38](#_Toc51145466)

[7.4. Sequence Diagram di dettaglio – Modifica parametri ambientali target 40](#_Toc51145467)

[7.5. Activity Diagram di dettaglio – Controlla parametri ambientali 40](#_Toc51145468)

# Introduzione

Il progetto che descriveremo di seguito rappresenta una singola iterazione di quattro settimane durante la quale sono stati ideate e realizzate alcune delle componenti del software da noi immaginato che ritenevamo colonne portanti dell’applicativo. In particolare il lavoro è stato temporalmente diviso in due stint tra i quali è intercorsa la pausa del mese di agosto:

* Una prima parte del progetto, principalmente composta dalle fasi di ideazione, specifica dei requisiti ed una primissima parte di elaborazione, è stata sviluppata a scaglioni durante le battute finali di Luglio, a cavallo tra un bagno ed un altro; abbiamo stimato questo contributo al progetto complessivo come il contributo di una settimana di iterazione.
* Il secondo stint invece, riguardante per lo più l’elaborazione dei casi d’uso sui quali ci siamo soffermati e l’implementazione degli stessi, è durato tre settimane tra l’1 ed il 21 Settembre, che sono esattamente le tre settimane rimanenti per concludere l’iterazione di quattro settimane.

La scelta dei requisiti da sviluppare è stata fatta di comune accordo anche sfruttando il gioco del **planning poker**: nello specifico, abbiamo deciso che ciascuno di noi dovesse produrre una piccola bozza in cui venivano specificati i casi d’uso del sistema, anche solo con un breve titolo, e che ad ognuna delle funzionalità venisse associata una priorità, una stima dello sforzo in termini di tempo speso ed un rischio di business, ossia quanto un errore commesso su tali funzionalità si sarebbe propagato sulle altre. Abbiamo poi raccolto le idee in un breve meeting di un’ora dal quale sono emersi i casi d’uso su cui concentrarci, che non a caso, al netto delle classiche funzionalità CRUD, sono per due terzi funzionalità da noi giudicate critiche ed a rischio alto, come si può vedere nel documento di visione.

Ci siamo basati su UP come modello di sviluppo, applicando alcune pratiche agili laddove ci risultasse più comodo: ad esempio, fare brevi riunioni giornaliere di non più di un’ora al mattino ci è servito per prendere le decisioni principali sul progetto, per le quali era necessaria la partecipazione di tutti, oltre che per aggiornarci continuamente sullo stato di avanzamento del progetto in modo da spostare le nostre risorse sull’attività più onerosa. Questo perché, per motivi di vicinanza, solo per due giorni ogni settimana è stato possibile vederci da vicino in modo da velocizzare il processo decisionale e lo sviluppo del software, mentre la restante parte del lavoro è stato svolto da remoto, utilizzando Teams per le riunioni giornaliere. Un’altra pratica spesso messa in atto è il pair programming ed il code review, in modo da velocizzare il progresso di debug e di sviluppo software. Anche per i diagrammi UML è stato seguito un approccio simile: ogni diagramma terminato veniva mostrato agli altri due componenti del gruppo, in modo da testarne la leggibilità.

Abbiamo preso in prestito dalle pratiche spesso adottate in SCRUM la buona abitudine di costruire un piccolo backlog dell’iterazione, utilizzando la piattaforma Trello; l’utilizzo di una board comune per la to-do list e per i dubbi delle singole persone è stato di grande aiuto per pianificare le giornate di lavoro.

Per quanto riguarda l’ambiente di sviluppo, la parte desktop del progetto è stata sviluppata in Java, utilizzando come IDE Eclipse aggiornato alla versione di giugno 2020; le interfacce sono state create con WindowBuilder, un plugin per lo stesso Eclipse. Per la parte del microcontrollore invece è stato adoperato ArduinoIDE.

La documentazione scritta in UML è stata prodotta utilizzando Visual Paradigm; per la sincronizzazione di documenti e progetti software è stata creata una repository apposita di GitHub, visitabile sul sito <https://github.com/marcobarlo/greenhouse>.

# Documento di visione per progetto Greenhouse

**a – Introduzione**

Lo scopo di questo documento è di raccogliere, analizzare e definire le esigenze ad alto livello di astrazione degli utenti e le feature del sistema.

Ciò che si vuole sviluppare è un sistema software per la gestione automatizzata di una serra, che verrà utilizzato dai dipendenti per la visualizzazione della distribuzione delle colture, la gestione dell’impianto idrico e la regolazione dei parametri ambientali fondamentali per la corretta crescita delle piante, quali l’umidità, la temperatura e l’intensità luminosa.

La serra è composta da più sezioni, ovvero dei capannoni fisicamente separati, e all’interno di ogni sezione sono presenti varie aree coltivate, disposte per file. Ogni area coltivata assume una determinata posizione numerata all’interno della fila ed è descritta da una estensione in mq e un tipo di terreno che indica quali sono le coltivazioni che si possono piantare. In ciascuna area coltivata può essere piantata una determinata coltivazione per la quale è importante conoscere lo stato di avanzamento della lavorazione, la data della prossima operazione, il tipo di coltivazione e una descrizione. Infine il software deve comprendere un meccanismo per notificare gli addetti ad una determinata coltivazione, in modo da ricevere in tempo reale aggiornamenti sullo stato di lavorazione o avvisi importanti.

Il software deve permettere ad un agronomo in primo luogo di modificare l’ambiente in cui si trovano le piante, permettendo di inserire i parametri target sia alla creazione dell’ambiente che in caso di modifica di uno o più di essi. Il sistema deve poi cercare di mantenere i parametri ambientali reali prossimi a quelli target a meno di una soglia, leggendo i valori ogni 10 minuti. Deve permettere ad un impiegato senza particolari permessi di ricercare una coltivazione specificando una parte o tutti i parametri della stessa, tra i quali sono necessari almeno un parametro tra tipo e sezione oppure la fila e posizione, e in seguito di visualizzarne ulteriori dettagli come la descrizione e i parametri ambientali target ed attuali. Deve poi consentire al direttore della serra di aggiungere, modificare, cancellare sia le coltivazioni che le aree coltivate all’interno delle sezioni e all’agronomo di gestire le coltivazioni, dopo aver mostrato una schermata con un quadro generale delle sezioni . Se si sta aggiungendo una coltivazione, il sistema ricerca una area disponibile libera all’interno di una determinata sezione in input e la mostra all’utente che deve poi confermare. Se si sta modificando la coltivazione si può modificare l’avanzamento dello stato di una coltivazione, e tutti gli interessati ricevono una notifica. Entrambi gli utenti devono poter anche ricercare autonomamente un’area libera nel momento in cui si pianta una coltivazione e bisogna decidere dove posizionarla. Infine il software deve permettere ad un responsabile del software di modificare, aggiungere e rimuovere utenti e modificarne i permessi all’interno dell’applicativo.

**b – Gli utenti**

Gli utenti principali del sistema sono l’agronomo, il direttore della serra, il contadino della serra e il software manager.

**b.a - Profili**

* Agronomo: un tipo di utente generalmente principiante che utilizzerà il sistema per impostare i parametri ambientali delle sezioni della serra e per gestire le coltivazioni.
* Direttore della serra: è un principiante avanzato, utilizzerà il sistema per svolgere eventuali operazioni in luogo dell’agronomo oppure per gestire le aree di coltivazione della serra.
* Contadino: utente principiante, interagirà col sistema unicamente per cercare l’ubicazione di una sezione nella serra o la presenza di una coltivazione.
* Software manager: è l’utente più esperto, utilizzerà l’applicativo per gestire gli utenti ed avviare il sistema software.

**b.b - Contesto d'uso**

Il sistema si baserà su una semplice interfaccia a cui i dipendenti potranno accedere utilizzando le macchine desktop disponibili in azienda. Meme sull’app.

**b.c - Obiettivi**

Ciascuno degli utenti individuati ha diverse esigenze, tra le quali annotiamo principalmente:

* Agronomo: impostare i parametri ambientali, gestire le coltivazioni, ricercare un’area di coltivazione libera, cercare una coltivazione, iscriversi alla notification list.
* Direttore: gestire le aree di coltivazione e le coltivazioni, ricercare un’area di coltivazione libera, cercare una coltivazione, iscriversi alla notification list.
* Operaio: cercare una coltivazione, iscriversi alla notification list.
* Software manager: gestire gli utenti, avviare il sistema.

**c – il sistema**

Il sistema sarà sviluppato come applicazione desktop. Dovrà interfacciarsi con numerose componenti hardware quali i diversi sensori per il monitoraggio dei parametri ambientali (termometro, sensore per l’umidità, amperometro per la stima dell’intensità luminosa (?)) e per la loro modifica (serpentine per il riscaldamento, deumidificatori smart e led). L’unica componente software con cui il sistema comunicherà invece è il DBMS per la gestione degli utenti, che però potrà essere salvato in locale (?).

**c.a - Caratterizzazione**

* Il sistema è diretto a tutto il personale della serra.
* Lo scopo del sistema è supportare tutte le attività di gestione della serra semplificabili mediante il supporto tecnologico
* Il sistema appartiene alla categoria delle applicazioni desktop.
* I benefici derivanti dall'uso sono principalmente derivanti dalla semplicità con cui i dipendenti della serra potranno accedere ai parametri ambientali e allo stato delle coltivazioni, evitando operazioni ripetitive oltre che l’onere del salvataggio dei dati sulle varie zone coltivate.

**d – riassunto delle funzionalità**

Riassumiamo in questa tabella i benefici che il sistema presenta con associate le funzionalità che permettono di ottenerli.

|  |  |
| --- | --- |
| Beneficio | Funzionalità |
| Regolazione e controllo automatizzato dei parametri ambientali | PG1, PG2 |
| Gestione semplificata del ciclo di coltivazione delle piante | PG3, PG4, PG5, PG6 |
| Sicurezza | PG7, PG8 |

**e – features del sistema**

Nella tabella in basso sono descritte tutte le feature scoperte riguardanti il sistema software. A ciascuna feature è associato un codice identificativo, un nome e tre indicatori che rappresentano la sua importanza (priorità), lo sforzo richiesto per la sua implementazione (sforzo) e le probabilità che qualcosa vada storto durante la fase di realizzazione (rischio).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Nome | Priorità | Sforzo | Rischio |
| PG1 | **Impostazione parametri ambientali** | **Critica** | **Medio** | **Alto** |
| PG2 | **Controllo parametri ambientali** | **Critica** | **Medio** | **Alto** |
| PG3 | Gestione aree di coltivazione | Critica | Basso | Basso |
| PG4 | Gestione coltivazioni | Critica | Basso | Basso |
| PG5 | **Ricerca aree libere e sezioni** | **Importante** | **Basso** | **Basso** |
| PG6 | Iscrizione alla notification list | Importante | Basso | Medio |
| PG7 | **Accesso autenticato al sistema** | **Critica** | **Basso** | **Basso** |
| PG8 | Gestione degli utenti | Importante | Medio | Basso |

La prima iterazione di 4 settimane prevede lo sviluppe delle funzionalità **PG1, PG2, PG5 e PG7**.

# Glossario di visione

|  |  |
| --- | --- |
| Termini | Definizione e informazioni |
| Direttore | Direttore della serra che si occupa del coordinamento di colture e personale. |
| Agronomo | Esperto delle coltivazioni in termini di quantità di risorse necessarie e di buona salute delle piante. |
| Giardiniere | Manovale di basso livello che si occupa di piantare e tutte le azioni manuali. |
| Sezione | Area della serra contenente diverse file di aree coltivabili, avente ognuna un certo numero di posizioni, a cui corrispondono le singole aree. |
| Area Coltivata | Zona in cui si può piantare una Coltivazione. |
| Parametri Ambientali | I parametri di irradianza, umidità del suolo e umidità e temperatura ambientali. |
| Irradianza | Densità di corrente termica trasmessa. |
| Controllore | Piattaforma hardware impiegata per la realizzazione del progetto; generalmente un microcontrollore a bassa potenza come una scheda Arduino. |
| Sensore | I componenti hardware impiegati per la misurazione dei parametri ambientali |
| Attuatore | L’hardware impiegato per la modifica dei parametri ambientali: ad esempio, una serpentina per il calore per alzare la temperatura o una luce led per aumentare l’irradianza fornita alla pianta |

**Termini Definizione e Informazioni Formato Regole di validazione**

Notification List Lista dei dipendenti che devono lavorare su una Coltivazione, e che devono essere notificati se necessario

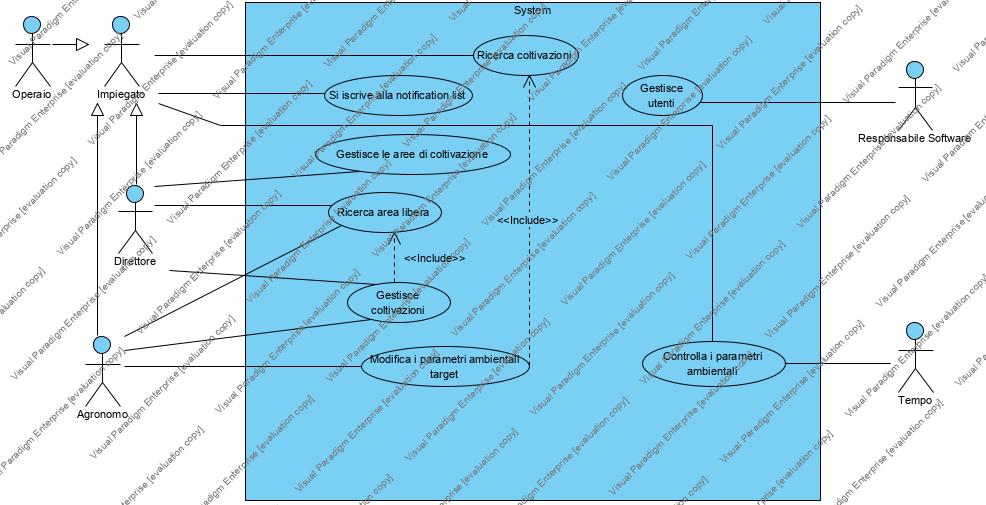
# Use Case model document

**Attori: direttore, operaio contadino[[1]](#footnote-1), agronomo esperto di piante, responsabile software, attore tempo**

1. Un impiegato **ricerca le coltivazioni**: Un qualunque impiegato deve poter ricercare le coltivazioni relative a una sezione della serra o di un certo tipo, sia all’interno di una data sezione che in tutte le sezioni. La ricerca può essere effettuata utilizzando come parametri di input il tipo di coltivazione, l’identificativo della sezione, la fila e la posizione della coltivazione, a patto che almeno uno tra il tipo di coltivazione e l’identificativo della sezione siano specificati. Il sistema restituisce la coltivazione o la lista di coltivazioni che corrispondono ai parametri. L’utente può scegliere di richiedere i dettagli della coltivazione in lista; in tal caso, il sistema mostrerà a schermo anche informazioni aggiuntive come lo stato della coltivazione ed i parametri ambientali.
2. L’agronomo **modifica i parametri ambientali target:** un agronomo della serra, previa autenticazione, deve poter scegliere la temperatura, l’umidità del suolo e l’illuminazione di ogni area coltivata. Il sistema mostra all’agronomo un quadro generale dei parametri delle varie sezioni. L’agronomo **ricerca una o più coltivazioni** ed in seguito inserisce almeno uno dei parametri ambientali da modificare, dopodiché il sistema verifica che i parametri in input siano plausibili, nel qual caso registra il cambiamento. L’impiegato riceve l'ok dal sistema del procedimento andato a buon fine.
3. Il direttore **gestisce le aree di coltivazione**: Il direttore della serra, previa autenticazione, deve poter gestire le sezioni della serra e le aree coltivabili al loro interno. Il sistema mostra un quadro generale delle varie sezioni, mostrando le aree coltivabili all’interno e le coltivazioni associate. In seguito, il direttore dà il comando al sistema di aggiungere/rimuovere/aggiornare dei parametri di alcune sezioni o di alcune aree all’interno, inserendo eventualmente i parametri necessari (come estensione, aree disponibili, tipo di terreno …??????). Il sistema verifica la plausibilità degli input e registra i cambiamenti.
4. Un agronomo o il direttore **gestiscono le coltivazioni:** Gli attori, previa autenticazione, devono poter gestire le varie coltivazioni della serra. Il sistema mostra il quadro generale delle coltivazioni e gli utenti danno ordine al sistema di creare/modificare/eliminare una o più coltivazioni. Se si sta aggiungendo una coltivazione, il sistema ricerca una area disponibile libera all’interno di una determinata sezione in input e la mostra all’attore che deve confermare. Se invece si stanno effettuando modifiche si può cambiare l’avanzamento dello stato di una coltivazione, generando l’invio di una notifica a tutti i dipendenti interessati. Il sistema registra i cambiamenti.
5. L’agronomo o il direttore **ricercano un’area libera:** Gli attori devono poter ricercare all’interno di una o più sezioni un’area libera che rispetti dei parametri. Gli attori possono inserire la sezione se vogliono limitare i risultati ad una sola sezione, altrimenti è sufficiente inserire l’area minima richiesta e la tipologia di terreno. Il sistema mostra i risultati della ricerca.
6. Un qualunque impiegato **si iscrive alla notification list** di una coltivazione: un impiegato della serra, previa autenticazione, deve poter segnalare la propria volontà di ricevere notifiche ogni qual volta si effettuino modifiche su una particolare coltivazione. L’impiegato **ricerca la coltivazione** e dà in input al sistema il comando per iscriversi alla notification list, o per essere rimosso se già presente. Il sistema registra i cambiamenti.
7. Il responsabile software **gestisce gli utenti**: Il software manager, previa autenticazione, dà in input al sistema il comando di aggiungere/rimuovere/modificare un utente, e dopo che il sistema richiede gli input il manager inserisce i dati in base a quelli necessari. Il sistema valida i dati in input e registra i cambiamenti.

**UC livello funzione**

1. Il responsabile software **avvia il sistema**: Il manager deve essere in grado di avviare in maniera semplice il software controllando che tutte le attività vengano effettuate correttamente e non ci siano errori. Dopo un fallimento del software il manager deve poterlo riavviare senza perdita di dati e in maniera sicura. Il sistema mostra l’andamento dell’avvio e mostra eventuali malfunzionamenti hardware/software.
2. Un attore si **autentica:** un attore di quelli della serra deve potersi autenticare, in base a ciò che richiedono le varie funzionalità, per validare il suo ruolo.
3. Periodicamente il sistema **controlla i parametri ambientali**: Ogni 10 minuti il sistema controlla che i parametri ambientali siano, a meno di tolleranza, coincidenti con quelli target, in caso contrario lancia un allarme che deve poter contenere le informazioni necessarie affinché gli impiegati possano controllare sul posto se c’è qualcosa che non va. Inoltre, se uno degli attuatori non risponde più o non invia più i dati al sistema, il sistema notifica (via mail) agli utenti interessati alla coltivazione il malfunzionamento.

****

Use Case Diagram

Stante il contesto applicativo, già in questa fase del progetto abbiamo previsto che il sistema si interfacci con diversi sensori e attuatori per svolgere i casi d’uso previsti. Tutto ciò è descritto nel diagramma di contesto in basso. Per essere più precisi, è già stata prevista un’interfaccia grafica per il generico impiegato (come vedremo più avanti nei wireframe sviluppati per le specifiche dei casi d’uso), oltre che un’interfaccia software per il collegamento del sistema con l’hardware previsto per il progetto.

Immagine che contiene testo, mappa

Descrizione generata automaticamente

Context Diagram

## 4.1. Ricerca una coltivazione

Ambito: Greenhouse application

Livello: user goal

Attore primario: un qualunque impiegato

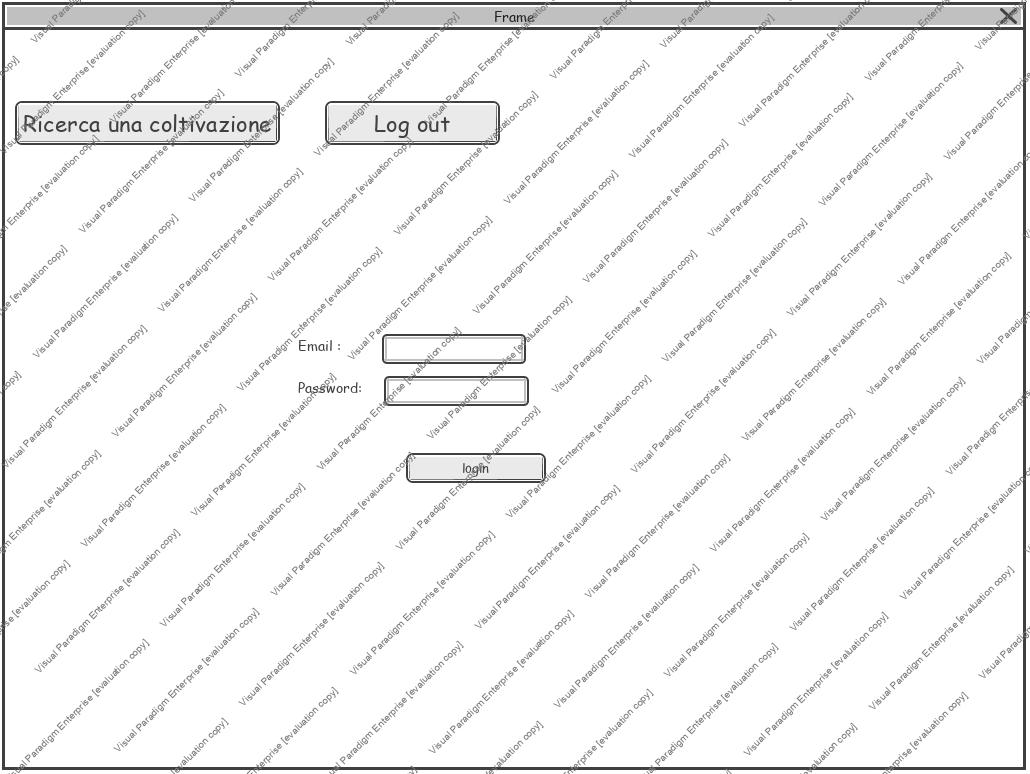
Stakeholders e interessi:

* L’impiegato: vuole sapere rapidamente quali coltivazioni sono presenti in una data sezione o scoprire la locazione di una specifica coltivazione e vederne i dettagli oppure per trovare i dettagli di una coltivazione di cui se ne conosce la posizione nella serra, ma non il tipo.
* Impresa della serra: vuole che gli impiegati riducano i tempi morti di ricerca all’interno della serra, per incrementare la produttività

Precondizioni: Sono presenti già delle coltivazioni all’interno delle sezioni

Scenario di successo:

1. L’impiegato dà il comando di ricerca di una coltivazione

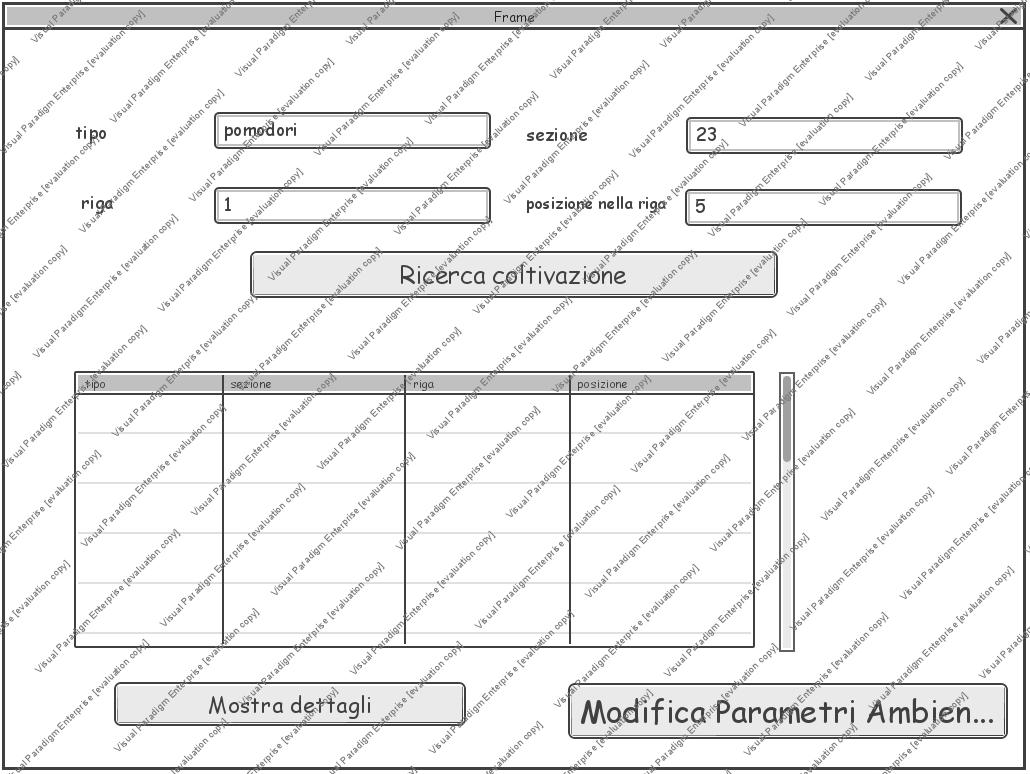


1. L’impiegato inserisce l’identificativo di una sezione della serra (numero positivo)
2. L’impiegato inserisce il tipo di coltivazione (numero positivo)
3. L’impiegato inserisce la riga della coltivazione nella sezione (numero positivo)
4. L’impiegato inserisce la posizione all’interno della riga della coltivazione (numero positivo)

Immagine che contiene disegnando

Descrizione generata automaticamente

1. L’impiegato dà l’ordine di ricercare secondo i parametri inseriti
2. Il sistema valida i dati di input
3. Il sistema restituisce una lista di tutte le coltivazioni che corrispondono ai criteri



Estensioni e scenari alternativi:

7a. Alcuni dei parametri non sono validi.

1. Il sistema segnala all’utente l’errore che alcuni parametri sono out-of-range oppure di tipo errato
2. Il sistema torna al punto 1 chiedendo all’utente di rieffettuare il reinserimento

7b. Non sono stati inseriti tutti i parametri.

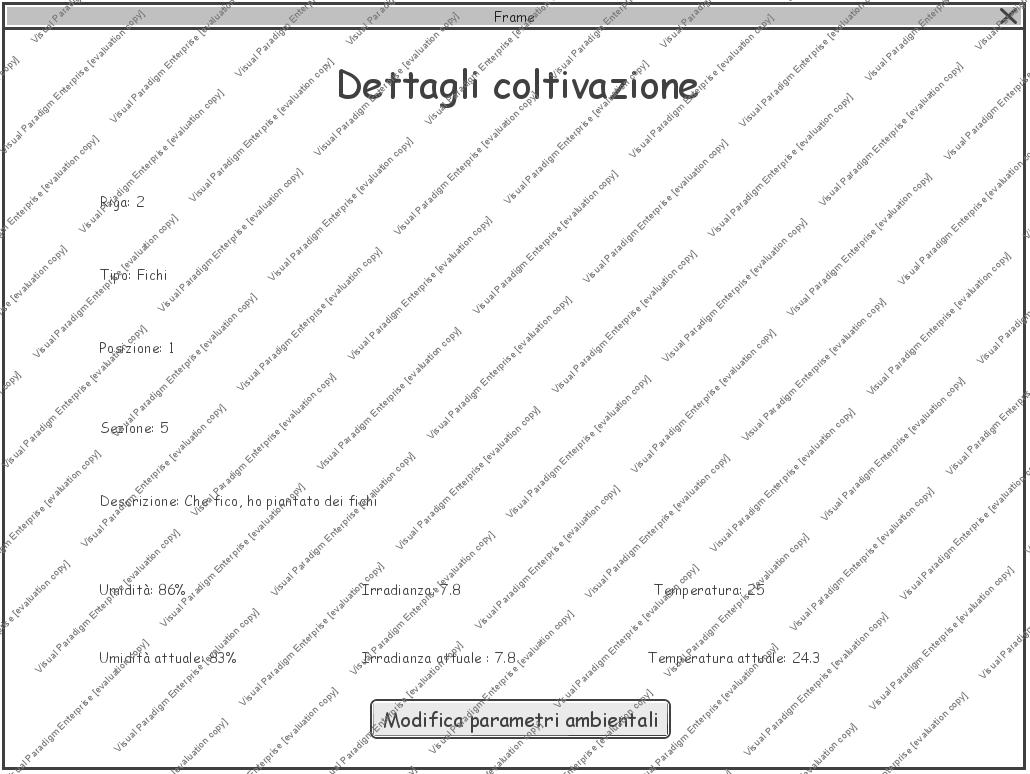
1. Il sistema effettua la ricerca solo in base agli altri parametri, non ponendo alcun filtro per i campi lasciati vuoti. Deve esserci almeno un parametro tra sezione e tipo e per inserire la posizione bisogna inserire anche la riga

7c Non è stato inserito alcun parametro

1. Il sistema segnala all’utente che bisogna inserire almeno un parametro di filtro tra sezione e tipo per effettuare una ricerca
2. Il sistema ritorna al punto 1

8a Il contadino vuole scegliere una specifica coltivazione

1. L’impiegato sceglie la coltivazione di interesse
2. L’impiegato chiede di mostrare i dettagli
3. Il sistema mostra i dettagli relativi a quella coltivazione inclusi parametri ambientali target e attuali



Il diagramma di sequenza di sistema risulta pertanto il seguente:

Immagine che contiene testo

Descrizione generata automaticamente

Ricerca Coltivazioni - SSD

## 4.2. Modifica i parametri ambientali target

Ambito: Greenhouse application

Livello: user goal

Attore primario: agronomo

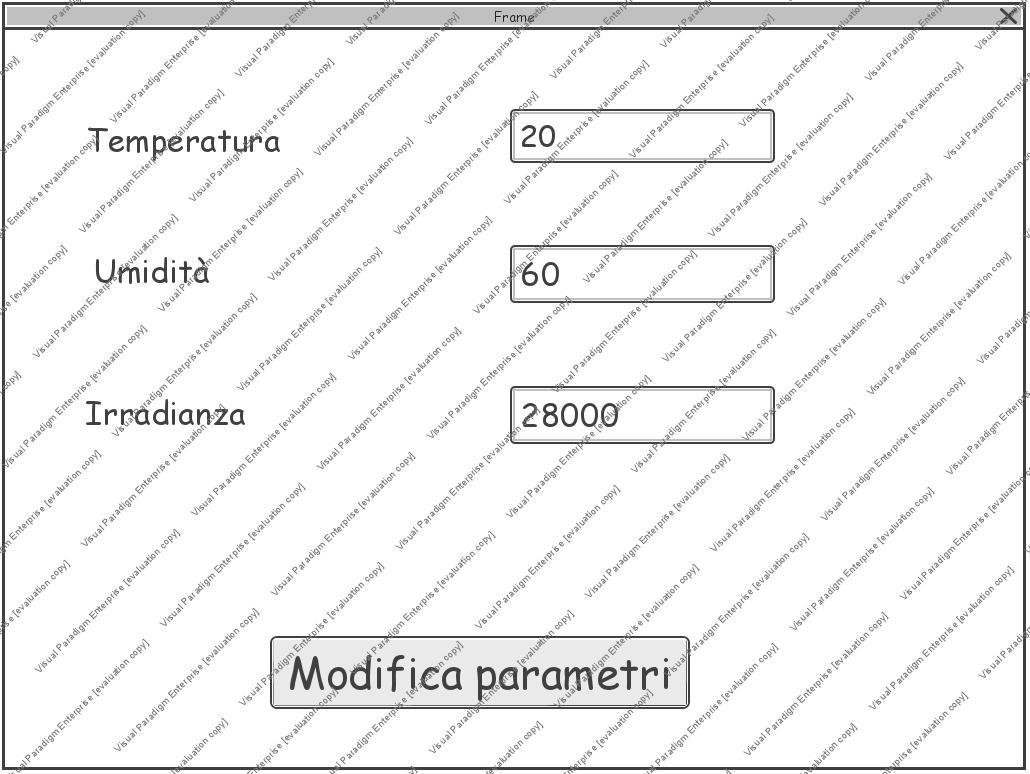
Stakeholders e interessi:

* Agronomo: vuole cambiare rapidamente i parametri ambientali di una sezione o di una area di una coltivazione per permettere una crescita migliore delle piante.
* Impresa della serra: vuole che le sue piante crescano al meglio e rapidamente per poter vendere prodotti di buona qualità. Vuole garanzie di funzionamento del sistema di controllo dei parametri.

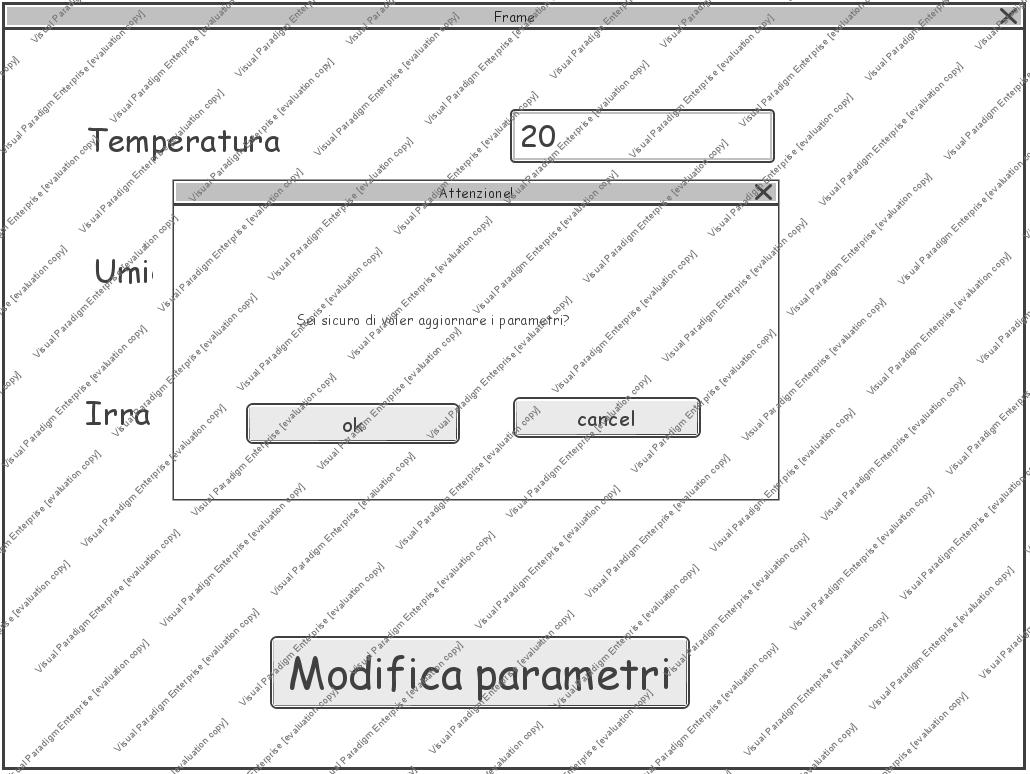
Precondizioni: l’agronomo si è autenticato, sono presenti già delle coltivazioni all’interno delle sezioni  
Postcondizioni: I dati ambientali target vengono registrati e il sistema ha attivato gli attuatori necessari a raggiungere i dati ambientali target, sulla base delle misure attuali.

Scenario di successo:

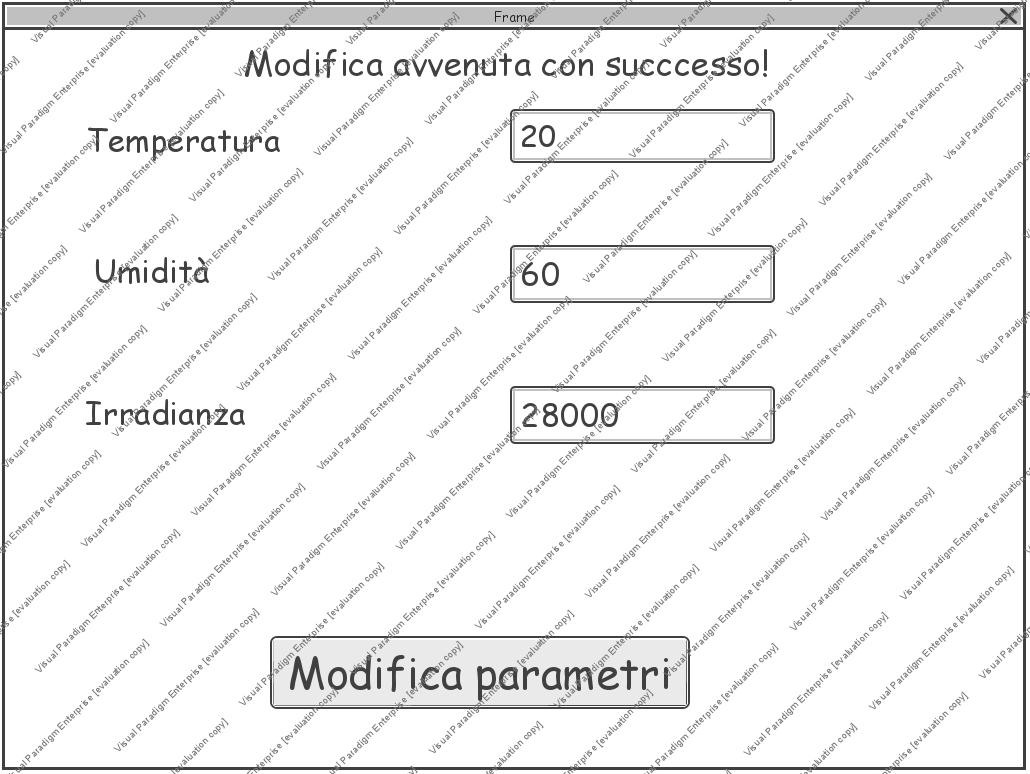
1. L’agronomo **ricerca una coltivazione**
2. L’agronomo sceglie una delle coltivazioni
3. L’agronomo dà in input al sistema il comando di modifica dei parametri ambientali
4. L’agronomo inserisce l’umidità del suolo, compresa tra 0 e 100
5. L’agronomo inserisce la temperatura ambientale, compresa tra -10 e 60 gradi
6. L’agronomo inserisce l’irradianza richiesta, compresa tra i 25k e i 100k lux



1. L’agronomo dà il comando di modifica dei parametri
2. Il sistema chiede conferma della volontà di modificare i dati, mostrando gli input



1. L’agronomo conferma
2. Il sistema valida gli input
3. Il sistema memorizza i dati ambientali
4. Il sistema comunica al controllore i nuovi parametri target
5. Il sistema dà conferma all’agronomo della corretta esecuzione di tutte le operazioni



Estensioni e scenari alternativi:

1a. Riferirsi al caso d’uso **ricerca una coltivazione**

2a. L’agronomo vuole modificare gli ambienti di più di una coltivazione

1. L’agronomo seleziona più di una coltivazione

3a. L’agronomo ha sbagliato a premere, non voleva modificare i parametri:

1. L’agronomo abortisce la modifica e torna al punto precedente

4-6a. L’agronomo cambia idea, non doveva modificare i parametri ambientali:

1. L’agronomo dà al sistema il comando di tornare alla schermata principale e annullare le modifiche in corso
2. Il sistema torna allo stato iniziale

7a. L’agronomo si accorge di aver sbagliato i parametri inseriti:

1. L’agronomo torna indietro e reinserisce i parametri (il sistema torna al passo 4)

9a. L’agronomo si accorge di aver sbagliato:

1. L’agronomo dà in input al sistema il comando di abortire le modifiche
2. Il sistema torna al punto 5

10a. I parametri richiesti non sono ottenibili con gli attuatori a disposizione

1. Il sistema segnala che il parametro impostato non è impostabile con gli attuatori a disposizione e chiede all’utente di reimpostare il parametro
2. Il sistema ritorna al passo 5

10b. Non sono stati inseriti tutti i parametri:

1. Il sistema considera i parametri non inseriti come non modificati, e quindi non li aggiorna

10c. Non è stato inserito alcun parametro:

1. Il sistema considera i parametri come invariati, quindi non effettua modifiche e termina lo scenario

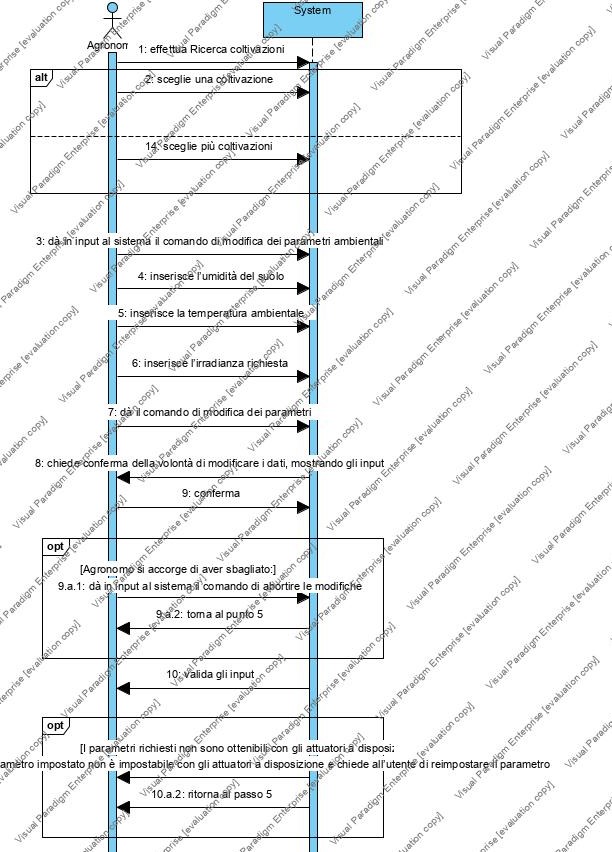
11a. Il sistema rileva un errore sulla memorizzazione

1. Il sistema segnala l’errore e segnala che bisogna eventualmente riprovare
2. L’utente dà al sistema un segnale di avvenuta lettura
3. Il sistema torna nello stato iniziale

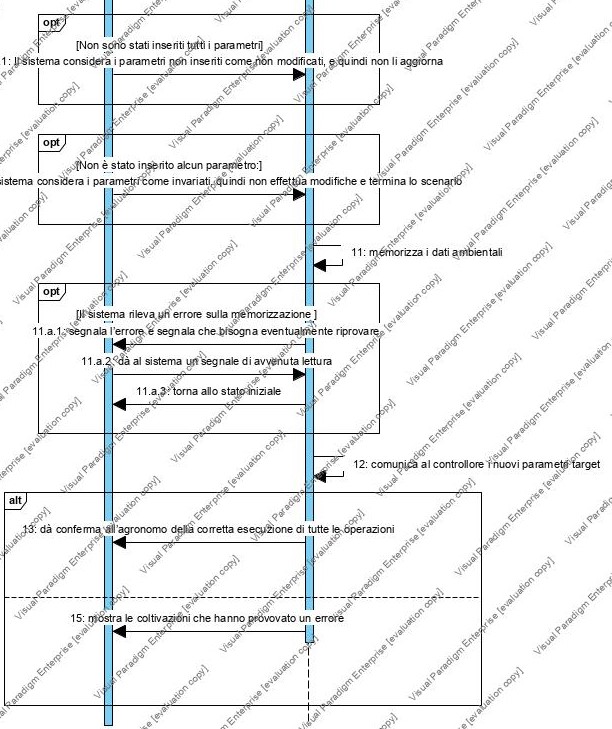
11b. Il sistema rileva un errore su una delle memorizzazioni

1. Il sistema elenca le coltivazioni per le quali c’è stato un errore

Di seguito forniamo il System Sequence Diagram corrispondente.



Modifica Parametri Ambientali – SSD – Parte 1



Modifica Parametri Ambientali – SSD – Parte 2

# Specifiche supplementari

In questa sezione del documento aggiungiamo tutti i requisiti supplementari non descritti dai casi d’uso.

**Funzionalità**

**Logging e gestione errore**

Il sistema mantiene un log permanente di tutti gli errori

**Sicurezza**

La maggior parte dei casi d’uso prevedono l’autenticazione degli utenti. Inoltre, il sistema in base all’utente autenticato deve fornire possibilità diverse di interazione.

**Fattori umani**

L’utente ha a disposizione un display all’interno della serra per poter interagire con il sistema. Il sistema deve fornire per interagire anche un touchscreen che possa essere attivato anche usando guanti da giardinaggio. (non implementato)

**Scalabilità**

Il sistema deve essere scalabile in modo da non gravare eccessivamente sul responsabile software in caso di necessità di sostituzione di un controllore, di un sensore o di un attuatore della serra.

**Affidabilità**

**Recuperabilità**

In caso di errore di comunicazione con la soluzione di log permanente dei dati prelevati dai sensori, il sistema dovrà salvare i dati in attesa del ripristino della comunicazione.

Se uno dei sensori non dovesse rispondere, i sensori necessiterebbero di un tentativo di riavvio. Qualora dovesse persistere l’errore si deve inviare una segnalazione per sostituire il sensore. (Necessitano di essere ridondati?)

**Adattabilità**

Il sistema deve essere capace di adattarsi, qualora si presentasse la necessità, di monitorare dati diversi e poter aggiungere un numero crescente di sensori.

**Configurabilità**

I microcontrollori vanno configurati mediante una procedura di startup demandata ad un programma principale. Inoltre, il sistema deve fornire la possibilità di aggiungere nuovi microcontrollori e sensori in caso di espansione della serra.

**Limiti hardware**

**Sensori**

Il sistema di controllo deve basarsi su piattaforme hardware a basso costo come le schede programmabili Arduino. Le misurazioni dei parametri ambientali sono effettuate con i sensori DHT11 per temperatura e umidità ambientale, il sensore igrometro TE215 per l’umidità del suolo; per la misura dell’irradianza invece va impiegato un fotoresistore analogico.

**Analisi dei costi**

Le scelte hardware siffatte permettono una realizzazione scalabile e a basso costo. In particolare, riportando i costi in maniera generale, si possono considerare i seguenti costi:

* N°1 macchina desktop equipaggiata con Java, la quale espone l’interfaccia all’utente
* N°1 broker MQTT deputato alla comunicazione tra la macchina Java ed i controllori mediante comunicazione Publish/Subscribe

Per ogni coltivazione sono inoltre necessari:

* N°1 microcontrollore a basso consumo (arduino UNO, NANO e simili)
* N°1 sensore per umidità e temperatura ambientale
* N°1 sensore per la rilevazione dell’irradianza
* N°1 sensore per l’umidità del suolo
* Serpentine per il riscaldamento
* Strisce led per aumentare la quantità di luce assorbita dalle piante
* Attuatore per l’irrigazione

**Comunicazione**

Per garantire la scalabilità del software, la comunicazione tra il controller ed il sistema principale è gestita mediante il protocollo di comunicazione MQTT ed il pattern publish/subscribe.

**Specifiche del sistema di controllo**

Il sistema di controllo ambientale deve essere realizzato con un controllore a ciclo chiuso e i parametri ambientali a regime non devono differenziarsi dai parametri target per più di un delta definito dall’utente. Inoltre il sistema di controllo per la temperatura e umidità deve andare a regime in due ore, quello per la luminosità in un minuto.

**Usabilità**

**Performance**

I sensori necessitano di avere tempi di risposta brevi, e il sistema deve poter essere attivo 24h. Il sistema deve elaborare in breve tempo per adattarsi ed effettuare il prima possibile qualsiasi azione necessaria per l’attivazione degli attuatori.

# Fase di elaborazione: diagrammi di alto livello

## Domain model

Contemporaneamente alla stesura dei casi d’uso, è stato necessario produrre un diagramma delle classi di alto livello, che riflettesse l’effettivo funzionamento della serra. Tra le considerazioni principali vi è innanzitutto la presenza di una classe contenitore, chiamata per l’appunto Serra, legata con un’associazione una a molti ad una classe Sezione; di lì in poi si dirama tutta la serra, con un’insieme di aree coltivate che se messe in coltura sono direttamente collegate ad una coltivazione; ciascuna coltivazione deve poi mantenere in un classe ambiente i proprio parametri ambientali target. Ogni ambiente è infine collegato ad una classe controllore secondo una relazione uno ad uno: ciascuna coltura, difatti, va controllata separatamente. È necessario specificare che per controllore, in questo caso, si intende l’oggetto software che applica la legge di controllo matematica per le coltivazioni. Ognuno di essi pertanto contiene al suo interno una serie di sensori ed attuatori, che a loro volta vengono specificati secondo una relazione gen-spec. La classe Descrizione Coltivazione è frutto dell’applicazione del pattern dell’abstraction occurrence sulla specifica coltivazione, in modo da non ridondare informazioni come il nome e la descrizione su coltivazioni della stessa natura, come pomodori o fichi: dunque se la classe Descrizione Coltivazione contiene informazioni sui fichi piantati nella serra, la classe Coltivazione mantiene informazioni sulla specifica pianta di fichi. Ciascuna coltivazione è poi associata ad una classe Impiegato, in modo che un qualunque impiegato possa ricevere eventuali aggiornamenti circa lo stato di una determinata coltivazione. Per motivi di semplicità di navigazione nel diagramma, abbiamo ritenuto opportuno anche collegare alla serra una classe detta Elenco descrizioni coltivazioni per recuperare più velocemente tali informazioni.

Immagine che contiene testo, mappa

Descrizione generata automaticamente

Domain Model

## Grasp model

Il passaggio successivo corrisponde ad un arricchimento del domain model con le responsabilità GRASP. Infatti, era necessario in questa fase determinare le responsabilità di creazione delle classi, ed eventualmente quelle di uso e di controllo. Le responsabilità di creazione sono risultate abbastanza ovvie: ad esempio, è la serra a dover creare la sezione, è la sezione a dover creare l’area coltivata, e così via. Il controllore crea poi le sue classi di appoggio, sensore ed attuatore. La serra inoltre deve necessariamente utilizzare la classe coltivazione per la risoluzione dei casi d’uso da noi previsti, in quanto ad esempio la ricerca delle coltivazioni è impossibile senza tale relazione d’uso. Infine, il controller desktop, da non confondere col controllore dell’ambiente, ha le responsabilità di controllo sulla serra, rappresentate dalla relazione “ricerca coltivazioni”.



Grasp Model

## Sequence Diagram di analisi – Ricerca coltivazioni

Così come evidenziato dai diagrammi di sequenza di sistema, il caso d’uso inizia con l’utente (in questo caso qualunque utente dato che per fare una semplice ricerca non sono richiesti particolari permessi) che attraverso l’interfaccia grafica, con cui comunica con il sistema, dà un ordine per ricercare le coltivazioni. Successivamente inserisce i parametri secondo i vincoli descritti dal caso d’uso (quindi deve inserire almeno un parametro tra tipo e sezione, e non può inserire la posizione nella riga senza inserire la riga) e dà il comando di ricerca. La UI se riscontra problemi sui dati di input mostra i relativi messaggi all’utente. La UI come prima cosa inoltra il comando a una classe Controller. Infatti dal momento che il sistema si prospetta sufficientemente grande, abbiamo previsto già qui delle classi fittizie di controllo che sappiano a chi inoltrare i comandi provenienti dall’utente e che sappiano orchestrare il flusso di controllo del caso d’uso. Il controller in questo caso inoltra la richiesta di ricerca alla Serra, che è l’entità che rappresenta l’oggetto root. La serra infatti, essendo unica e avendo consapevolezza delle varie sezioni, rappresenta il punto di ingresso ideale per il modello. La serra, essendo information expert delle sezioni, interroga le sezioni (o la sezione, se ne è stata inserita una) delegando loro la ricerca; a loro volta, le sezioni sono information expert delle aree coltivate e possono così trovare le aree con fila e posizioni corrispondenti ai parametri. A questo punto non rimane che chiedere alle aree coltivate che corrispondono ai parametri quali siano le coltivazioni piantate in esse, se ne hanno una. Il controller ora ha una lista delle coltivazioni corrispondenti ai criteri di ricerca, e interrogando le coltivazioni può ottenere le loro informazioni.

A questo punto se l’utente ordina all’interfaccia di mostrare i dettagli di una coltivazione, l’interfaccia inoltra nuovamente l’ordine al controller, che interrogando la coltivazione, attraverso delegazioni (verso l’ambiente, ad esempio) può ottenere le ulteriori informazioni, come i parametri ambientali target, che prima non era possibile visualizzare.

Immagine che contiene testo, mappa

Descrizione generata automaticamente

Ricerca Coltivazioni Sequence DIagram

## Sequence Diagram di analisi – Modifica parametri ambientali target

Dopo aver ricercato le coltivazioni, se ne possono modificare i parametri ambientali target, di una o più coltivazioni alla volta.

L’utente (agronomo, dal momento che solo lui ha i permessi per modificare i parametri ambientali) dà l’ordine all’interfaccia di modificare i parametri target. A questo punto deve inserire i parametri ambientali (tutti o in parte, purché ne inserisca almeno uno altrimenti resta tutto invariato). L’interfaccia deve validare i valori che devono essere compresi nei range specifici dettati dai casi d’uso. Una volta inseriti valori corretti e una volta dato il comando di effettuare la modifica (dopo conferma, dato che è una operazione critica), per ogni coltivazione da modificare l’interfaccia inoltra il comando al controller, che interroga gli oggetti coltivazione per modificarli. Ogni coltivazione, conoscendo il suo ambiente può modificare i parametri target dell’ambiente, ma solo dopo aver dato ordine al controllore (questa volta inteso non in senso di classe fittizia che serve come primo oggetto contattato dall’interfaccia bensì di oggetto che applica la legge di controllo matematica per gli ambienti) e aver ricevuto l’ok di avvenuta modifica. Se il salvataggio dei parametri modificato non andasse a buon fine, l’interfaccia segnala all’utente che c’è stato un errore nel salvataggio dei parametri, quindi magari l’utente deve ritentare.

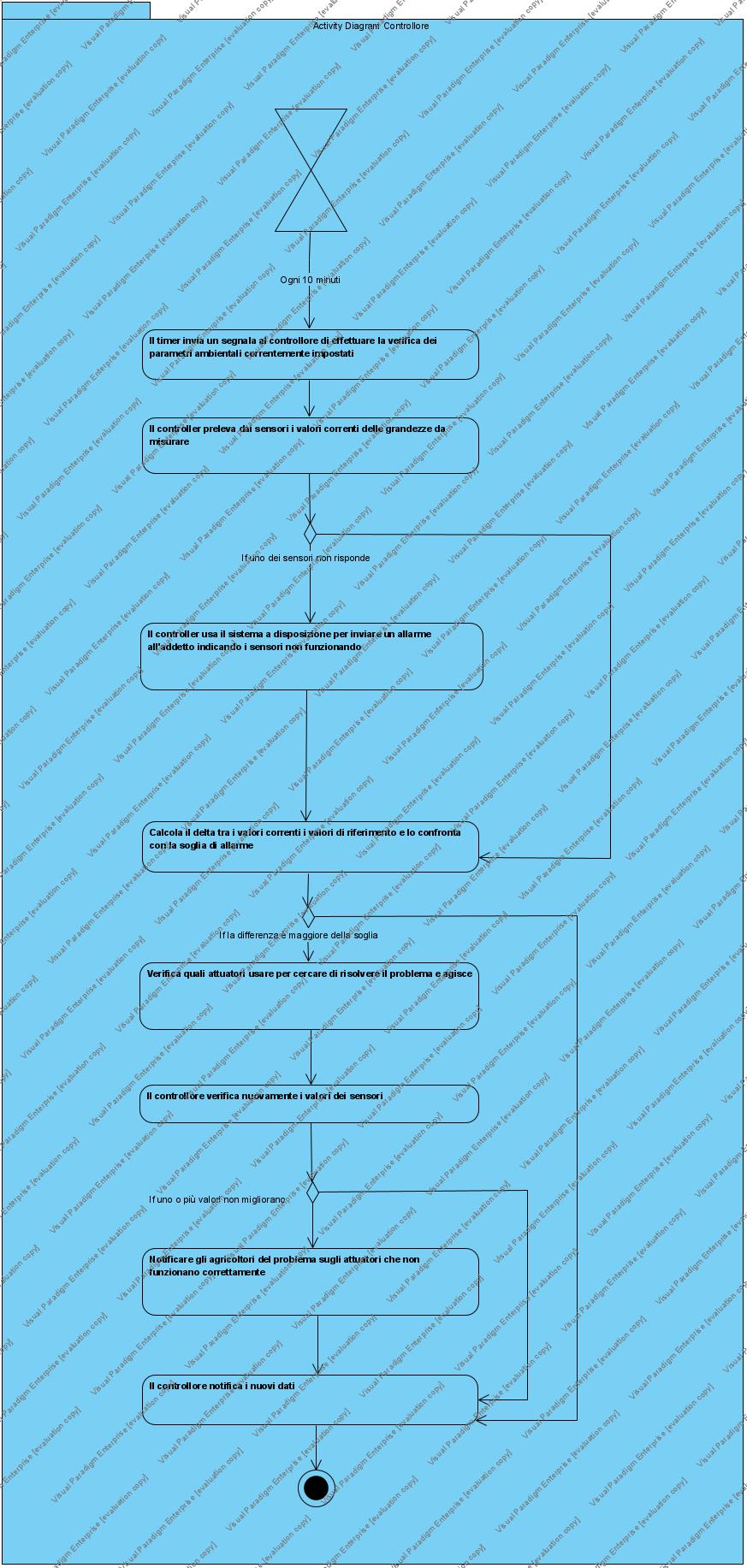
Immagine che contiene testo, mappa

Descrizione generata automaticamente

Modifica parametri ambientali target Sequence DIagram

## Activity diagram di analisi – Controlla parametri ambientali

Passiamo ora al funzionamento del sistema di controllo dei parametri ambientali. Seguendo un evento temporizzato con una cadenza di dieci minuti, il sistema deve verificare che i parametri ambientali siano ancora vicini ai parametri target definiti dall’utente. In caso affermativo, il controllore non fa nulla. Altrimenti, qualora un sensore non funzionasse o uno dei parametri ambientali superasse la soglia di tolleranza entro la quale dovrebbe trovarsi rispetto al target previsto, il controllore manderà un messaggio di errore. Il flusso di controllo è descritto ad alto livello dal seguente diagramma di attività.



Controlla parametri ambientali – Activity Diagram

# Architettura del sistema

## Overview dell’architettura

Per soddisfare i requisiti necessari in questa iterazione è stata scelta un’architettura con un componente software in Java, in modo da poter sfruttare la portabilità e la semplicità di tale linguaggio, ideata per essere installata su un PC desktop. Un altro componente è invece quello di controllo vero e proprio, che ha le responsabilità di mantenere i parametri ambientali vicini a quelli target a meno di una soglia di errore.  
Questo componente è stato sviluppato in C/C++, dal momento che tale linguaggio è più adatto ai sistemi hard real-time e firm real-time a causa della sua predicibilità, oltre ad essere meno *resource consuming* dal momento che i microcontrollori hanno solitamente hardware molto più ristretto dei comuni PC. In particolare per lo sviluppo del componente sono state utilizzate le librerie Arduino, presupponendo controllori di questa famiglia nel deploy, per la rapidità di scrittura del software, per l’ottima portabilità e scalabità (cambiando microcontrollore Arduino il codice sorgente può essere portato, a meno di limitazioni hardware, *as-is*).

Dati i compiti estremamente diversi dei due componenti sopra descritti, e data anche la loro eterogeneità, sia come hardware sia come software, assume una importanza critica il connettore che permetta ai due componenti di comunicare. Per la comunicazione è stato scelto il protocollo MQTT 3.1, con broker Mosquitto. MQTT si può dire quasi uno standard de facto nell’ambiente IoT. Permette infatti una comunicazione leggera ma soprattutto scalabile, pensata per occupare poche risorse e poca banda, ma che può garantire anche robustezza in base ai vari livelli di Quality of Service QoS. Con questa scelta abbiamo fatto in modo che i componenti fossero disaccoppiati a livello di sincronizzazione e spaziale (il broker per il momento è in locale ma potrebbe anche cambiare locazione). In tal modo se uno dei componenti fallisse, il resto del software continuerebbe a funzionare data la natura asincrona della comunicazione messa in piedi. Abbiamo ritenuto questa motivazione fondamentale, dal momento che soprattutto se il software del PC desktop fallisse improvvisamente, i controllori dovrebbero comunque garantire i giusti parametri ambientali per la sopravvivenza delle piante. Per lo stesso motivo è stata scartata l’idea di dare la responsabilità della legge di controllo al software desktop, senza considerare una considerevole latenza tra la lettura dei dati e l’attuazione della legge di controllo. Inoltre, il protocollo MQTT garantisce anche una certa flessibilità e scalabilità poiché se i requisiti cambiassero, basterebbe aggiungere/rimuovere o modificare dei topic, si potrebbe cambiare l’hardware e di conseguenza il software di uno o più microcontrollori lasciando il resto inalterato, e i microcontrollori potrebbero comunicare sia in maniera wired che wireless tramite lo stack TCP/IP.

Per utilizzare il brokerr MQTT sono state necessarie le librerie PubSubClient dal lato Arduino, e le librerie Paho dal lato Java.

Ultimo componente dell’architettura è il Database che garantisce la persistenza dei dati, al quale il componente software in Java si connette (tramite Hibernate) con le JDBC. E’ stato scelto MariaDB poiché leggero e semplice da utilizzare, soprattutto data la perfetta integrazione col tool di Visual Paradigm per Hibernate.

L’architettura del sistema è efficacemente riassunta dai diagrammi dei componenti che proponiamo in basso: nello specifico, il primo diagramma mostra la connessione tra il Mainframe ed il componente Controller con uno stereotipo chiamato MQTT. Il secondo diagramma dettaglia questa connessione, mostrando l’interazione col broker Mosquitto.

Immagine che contiene testo

Descrizione generata automaticamente

Component diagram

Immagine che contiene mappa, testo

Descrizione generata automaticamente

Component diagram – MQTT specification

In ultimo, mostriamo con un diagramma dei package l’architettura complessiva del sistema.



Package Diagram

## Componente MainFrame

Scendendo nel dettaglio dell’architettura interna dei singoli componenti spieghiamo le scelte architetturali del componente Mainframe.

E’ stata scelta una architettura a layer per diversi fattori, dei quali sicuramente la natura transazionale del software è stato quello preponderante; inoltre, tale architettura garantisce scalabilità ,modificabilità, sicurezza e comprensione del software grazie ai livelli crescenti di astrazione. Dal momento che non sono state fatte richieste specifiche sulle performance, abbiamo scelto di applicare uno stile Strict Layer (o Closed Layer), per avere un software di maggiore qualità. Con questa scelta ogni layer si appoggia solo al layer immediatamente sottostante, difatti facendo da client rispetto alle funzioni offerte.  
I layer individuati hanno i compiti funzionali classici di interfaccia video con l’utente (Layer UI), controllo, comunicazione e orchestrazione dei workflow complessi (Layer Business Logic), e infine rappresentazione e modifica del modello dei dati (Layer Model).

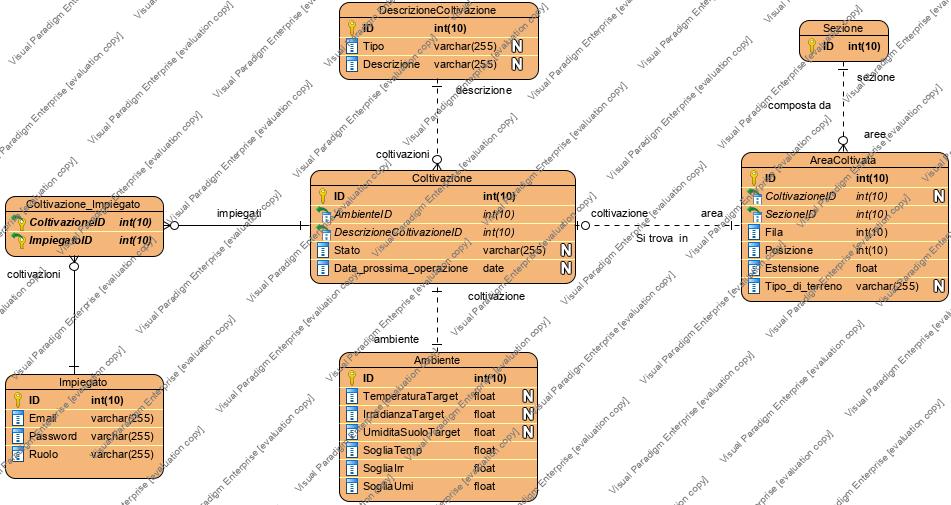
In seguito, dal momento in cui è sorta la necessità di aggiornare alcune schermate in base all’arrivo di particolari eventi e di notificare la stessa mediante allarmi provenienti dal lato dei microcontrollori, è stato ritenuto necessario un bus di eventi trasversale per trasportare “verso l’alto” alcune informazioni, senza violare il vincolo dello stile a layer di non poter comunicare coi layer superiori, rendendo di fatto l’architettura un ibrido di vari stili. La propagazione di eventi verso il basso per il momento non è vietata, tuttavia è fortemente sconsigliata. Scelta importante è stata anche quella di isolare le responsabilità della connessione col broker MQTT in una classe apposita, chiamata Connection, appartenente al package di connessione, in cui ci sono anche i messaggi che vengono pubblicati sul bus, in maniera da esporre interfacce per la comunicazione del tutto generiche e quindi assicurare una maggiore comprensibilità e manutenibilità del codice, oltre che modificabilità nel caso in cui la comunicazione in futuro non avvenga più tramite MQTT o nel caso in cui il client cambiasse piattaforma. Il package connection include al suo interno anche la logica dei messaggi di errore: avendo infatti su di sé le responsabilità di comunicazione, è risultato più semplice demandare la creazione dei messaggi di errore direttamente al package di connessione, mediante la creazione di una classe MexAllarme, derivata poi in altre sottoclassi specifiche per allarme (Allarme1, Allarme 2 etc.).

Per quanto riguarda l’interno dei vari layer evidenziamo la scelta di mantenere delle classi vuote (come nella UI o nella Business Logic) che hanno il solo scopo di dare una visione di insieme del software includendo classi non ancora implementate che riguardano requisiti da sviluppare in eventuali iterazioni future. In tal modo si ha già una struttura complessiva seppur molto approssimata del software, utile per prendere decisioni in ottica di cambiamenti futuri in maniera più semplice.

Da sottolineare anche la scelta di applicare un pattern facade al controllo per dare al layer un unico punto di ingresso, in maniera da rendere il codice facilmente modificabile qualora fosse necessaria una ristrutturazione del layer. In realtà oltre al facade lo strato denominato “Business\_Logic” espone anche due classi (DettagliBusiness e ColtivazioneBusiness) che portano varie informazioni al loro interno, appartenenti a più di una classe del modello, e servono a riassumere informazioni da esporre poi ai layer superiori. In tal modo lo strato di controllo non restituisce classi di modello all’interfaccia, poiché violerebbe il vincolo di strict layer, ma classi del livello di Business Logic. È importante sottolineare che tutti i package che si interfacciano con quello di Business Logic interagiscono con esso solo tramite i punti di ingresso messi a disposizione del package, quindi anche quello di connessione accede al layer di Business Logic solo attraverso il facade e le classi di ritorno esposte.

All’interno del modello è possibile trovare le classi presenti già nel modello GRASP, ma solo fino alla classe Ambiente, dato che le altre sono logicamente contenute nell’altro componente. La classe ambiente attuale è una classe di raccordo con il componente di controllo e ha lo scopo di mantenere in memoria i dati in arrivo dai controllori.

Il componente comunica col database tramite delle funzioni inserite nelle classi di persistenza, autogenerate dal framework di persistenza Hibernate. Il codice, che comprende anche classi aggiuntive, è stato autogenerato tramite un wizard integrato in Visual Paradigm, che si occupa anche di creare lo schema per il database, e altri script di utilità e file di DDL. Mediante il wizard vi erano diverse modalità per generare le classi di persistenza, ossia quelle poi presenti nel modello ERD. La nostra scelta è ricaduta sul modello a classi statiche, in cui ogni classe di persistenza, oltre a racchiudere i dati e alcune funzioni di utilità, espone anche le funzioni autogenerate per le chiamate a database. Per le collezioni è stata scelta la modalità di caricamento a lazy loading, per evitare di gravare eccessivamente sul database in caso di caricamenti frequenti: col lazy loading, le collezioni vengono caricate in memoria solo se strettamente necessario, diminuendo il traffico di dati.



ERD Model

ORM DIAGRAM. ERD diagram autogenerato con id autocreati (interi ad autoincremento)….perché i valori target sono nullable???

Il tool di VP produce codice Java in cui, per ciascuna classe di persistenza, genera le seguenti classi aggiuntive:

* Criteria per le query a database composte; i *Criteria* sono delle classi che incapsulano la logica di richiesta a DB: in questo modo non è necessario scrivere a mano la query SQL, bensì è sufficiente utilizzare i metodi esposti dalla classe Criteria.
* DetachedCriteria per effettuare query in particolari condizioni, ossia senza dover creare una sessione per le query
* SetCollection come containers solo per le classi con aggregazioni uno a molti

In ultimo, il tool ha generato un file .jar che espone tutte le funzioni necessarie al corretto funzionamento delle classi create, oltre che eventuali metodi utili per l’utente (come il metodo getByOrmId() per le query a database semplici data la chiave primaria), oltre che alcuni file .java per la semplificazione della creazione, distruzione e popolamento del database e dei file .xml di configurazione che salvano al loro interno lo schema del database stesso.

La nomenclatura è identica per tutte le classi: ad esempio, per la classe coltivazione esistono le classi ColtivazioneCriteria e ColtivazioneDetachedCriteria. Essendo l’associazione con DescrizioneColtivazione del tipo una a molti, la classe DescrizioneColtivazione ha un container che è ColtivazioneSetCollection.

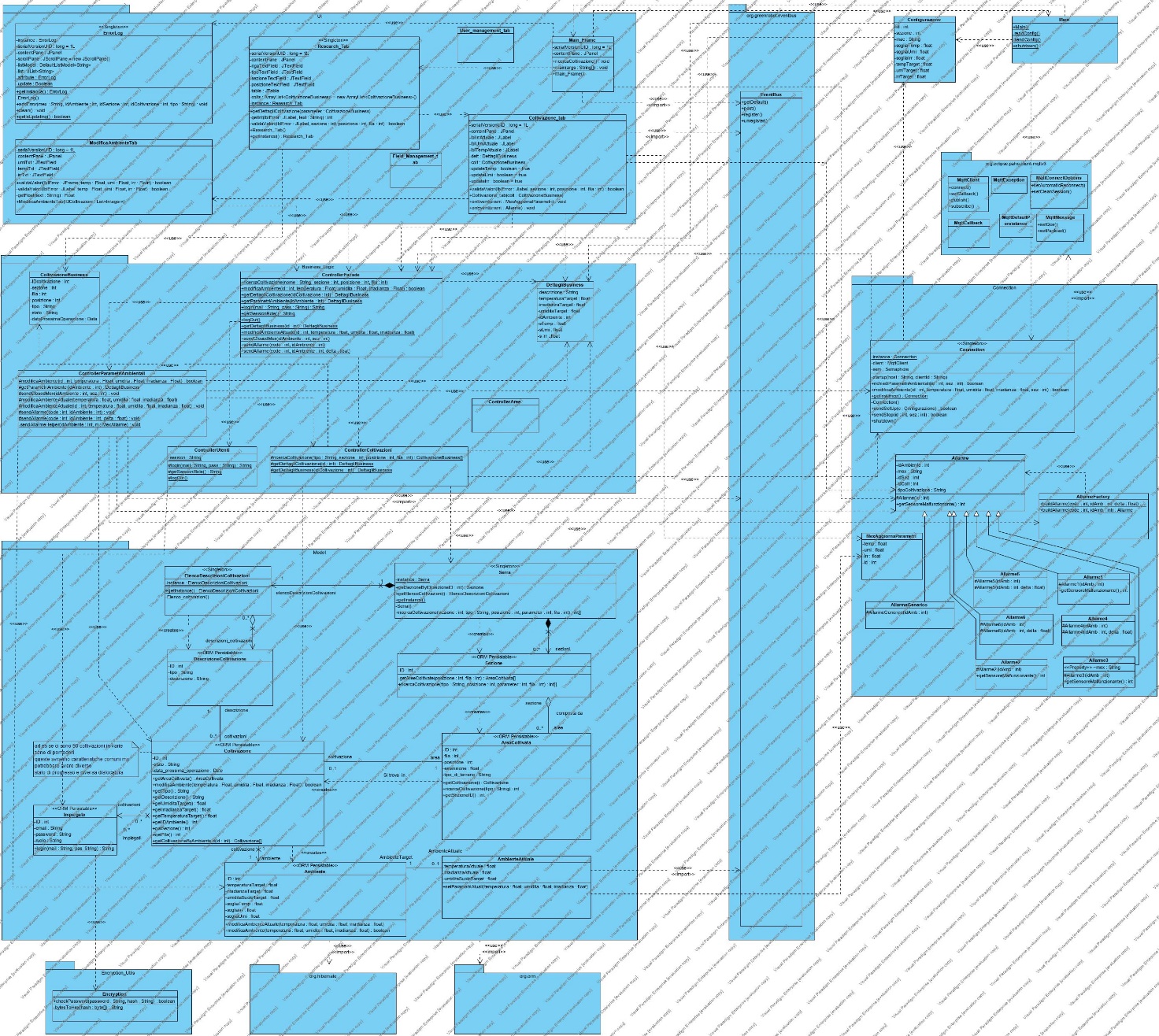
Immagine che contiene testo

Descrizione generata automaticamente

ORM Diagram

Un ulteriore package chiamato “Encryption\_Utils” è stato inoltre aggiunto in un secondo momento, quando è stata messa a punto la funzionalità per il login dell’utente. Il sistema, seppur abbastanza semplice, prevede infatti il salvataggio in database dell’hash della password dell’utente. Al login poi il sistema critta la password inserita, verifica che i due hash coincidono ed in caso affermativo garantisce l’ingresso nel sistema all’utente. Tale package è stato necessario perché la responsabilità di cifratura e verifica delle password non poteva essere attribuita a nessuno degli altri strati poteva per diversi motivi di business; dunque è stato necessario creare l’utilità Encryption\_Utils, collocata logicamente in un layer sottostante al Model.

Nel componente Mainframe sono infine presenti anche altre due classi (Main e configurazione) che sono fuori dai package perché non hanno una coesione logico/funzionale con altri elementi, bensì incapsulano la logica di start-up, configurazione e shutdown del software, secondo una coesione di tipo temporale.



Package Diagram Desktop Component

## 7.3. Componente ControlUnit

Il componente di controllo invece è strutturato secondo il pattern del **Compute-Sense-Control**. La scelta di questo pattern è quasi obbligata nel momento in cui è necessario interagire con l’ambiente esterno tramite sensori e attuatori di vario genere, come è possibile osservare nel diagramma di contesto. Data la disponibilità del linguaggio C++ su Arduino il componente è stato strutturato con uno stile OO, scelta non banale perché su altri microcontrollori con IDE diversi questo non è possibile. La classe di controllo si occupa di eseguire il loop di controllo periodicamente, leggendo dai sensori e scrivendo sugli attuatori. I sensori e gli attuatori specifici, come il sensore di umidità e temperatura o la serpentina riscaldante, sono legati da costrutti Gen-Spec con le classi padre sensore e attuatore. Il loro compito è esporre un’interfaccia unificata per l’interazione col controllore, che in C++ significa implementazione di funzioni virtuali ed eredità da classi astratte). Anche in questo componente le responsabilità di comunicazione sono state date ad una singola classe per gli stessi motivi del componente MainFrame. È presente inoltre la classe ambiente target che rappresenta i parametri target per il controllore.

Immagine che contiene testo

Descrizione generata automaticamente

Package Diagram ControlUnit

### 7.3.1. Comunicazione

Data l’eterogeneità dei componenti sviluppati è stato possibile sviluppare MainFrame e ControlUnit separatamente, dopo aver deciso però le modalità di comunicazione tra i due. In particolare, è stato necessario stabilire a priori quali fossero i topic di comunicazione e quali fossero i formati dei messaggi inviati, dato che in MQTT il payload non è altro che un array di byte.

I topic stabiliti sono stati i seguenti :

* “GH/SetUp” per i pacchetti di configurazione da inviare ai controllori.
* “GH/Dati” per i dati ambientali che il controllore invia al sistema desktop.
* “GH/Errore” per gli allarmi e le situazioni di errore del controllore.
* “GH/”+numero della sezione+”/cmd/STROBS” per i pacchetti di richiesta di lettura dati. Vengono inviati dal MainFrame a tutti i controller appartenenti alla sezione della coltivazione di cui l’utente vuole conoscere i dettagli. L’id dell’ambiente da cui recuperare i dati, e quindi del controller corretto, è contenuto nel payload del pacchetto; ciascun controller, pertanto, accetta il pacchetto solo se il suo ID coincide con quello nel payload, altrimenti lo scarta.
* “GH/”+numero della sezione+”/cmd/STPOBS” per i pacchetti di stop lettura dati, ossia quando l’utente non vuole più osservare i parametri ambientali, e quindi il controller non deve più inviarli. Anche qui l’id corrispondente al controller giusto è presente nel payload.
* “GH/”+numero della sezione+”/cmd/Mod” per i pacchetti di modifica dei parametri target.
* “GH/Ack” per i pacchetti di acknowledgment della richiesta di modifiche parametri ambientali.

Il numero della sezione utilizzato come topic è stato previsto in quanto abbiamo ritenuto che in futuro possa essere utile poter mandare messaggi a tutta una sezione, anche perché riteniamo plausibile che in una sezione si possano trovare coltivazioni affini che vengano controllate in maniera simile.

Per quanto riguarda il formato dei pacchetti:

* Un pacchetto di SetUp contiene in ordine il MAC address del dispositivo da configurare, l’id dell’ambiente da controllare (signed int 4 byte), l’id della sezione in cui si trova (signed int 4 byte), i valori target dell’ambiente (float 4 byte)(in ordine temperatura, umidità, irradianza), i valori delle soglie di tolleranza per il controllore (float 4 byte) (seguendo lo stesso ordine dei valori ambientali).
* Un pacchetto dati invece contiene in ordine l’ID dell’ambiente controllato (signed int 4 byte) e in seguito i tre valori ambientali letti sempre nel consueto ordine (float 4 byte).
* Un pacchetto di errore contiene sempre l’id dell’ambiente e un intero che rappresenta il codice di errore.
* Un pacchetto di richiesta dei parametri ambientali contiene solo l’ID dell’ambiente interessato, così come un pacchetto di stop e uno di ack.
* Un pacchetto di modifica parametri ambientali contiene invece in ordine l’ID dell’ambiente e i tre float che rappresentano i valori target nel consueto ordine.

Il flusso di funzionamento del programma è approssimativamente questo:

In fase di startup il software desktop legge da un file di configurazione che mantiene le associazioni tra MAC address, ID coltivazioni e ID sezione, di cui quest’ ultimo non viene prelevato da database per questioni di efficienza all’avvio. In seguito preleva le altre informazioni come valori target e soglie dal database e invia i pacchetti di configurazione per ogni entry del file di configurazione. I microcontrollori partono col loro ciclo di controllo solo dopo aver ricevuto tale pacchetto. Mostriamo nel diagramma di sequenza sottostante il funzionamento dettagliato della fase di startup.

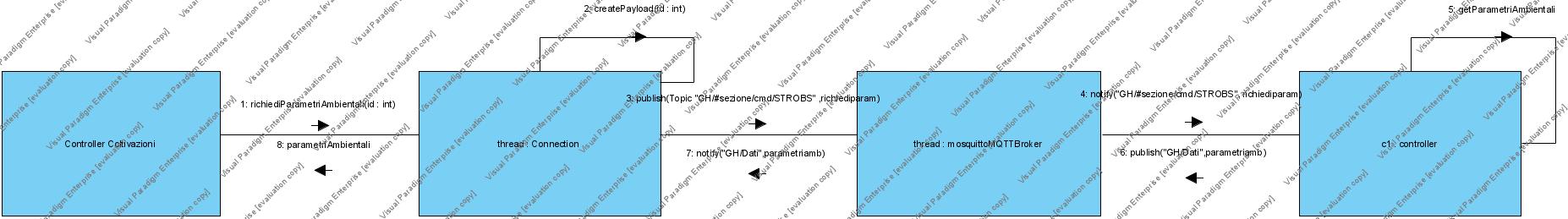
Immagine che contiene testo, mappa

Descrizione generata automaticamente

Startup Sequence Diagram

In ogni momento un controllore che rilevi un problema (per ora ci sono 6 codici di errore, 3 per sensore mal funzionante, 3 per attuatore mal funzionante), può mandare un allarme sul topic apposito, e il software desktop mostrerà a video il messaggio di allarme.

Durante il runtime del software desktop, quando l’utente apre una schermata di dettaglio di una coltivazione, il software manda un pacchetto di richiesta lettura con l’ID dell’ambiente relativo, che corrisponde al topic di STROBS. Il flusso di comunicazione è descritto nel diagramma sottostante.



Richiedi parametri ambientali Communication Diagram

A questo punto il controllore addetto legge i dati, manda un pacchetto di risposta Dati e si pone nello stato di osservato. Questo vuol dire che ad ogni ciclo di controllo, alla lettura dei dati questi saranno spediti al componente desktop, fin quando non arriva un pacchetto di stop, che pone il controllore nello stato di non osservato. Questo metodo è stato escogitato per evitare che i controllori mandassero dati a prescindere dal corretto funzionamento dei sensori, rischiando di saturare la banda disponibile per la comunicazione in caso di un alto numero di controllori e inviando pacchetti che di fatto non servirebbero a nessuno. Con questo pacchetto dati il componente desktop può aggiornare l’interfaccia con i dati appena letti.

Immagine che contiene testo, mappa

Descrizione generata automaticamente

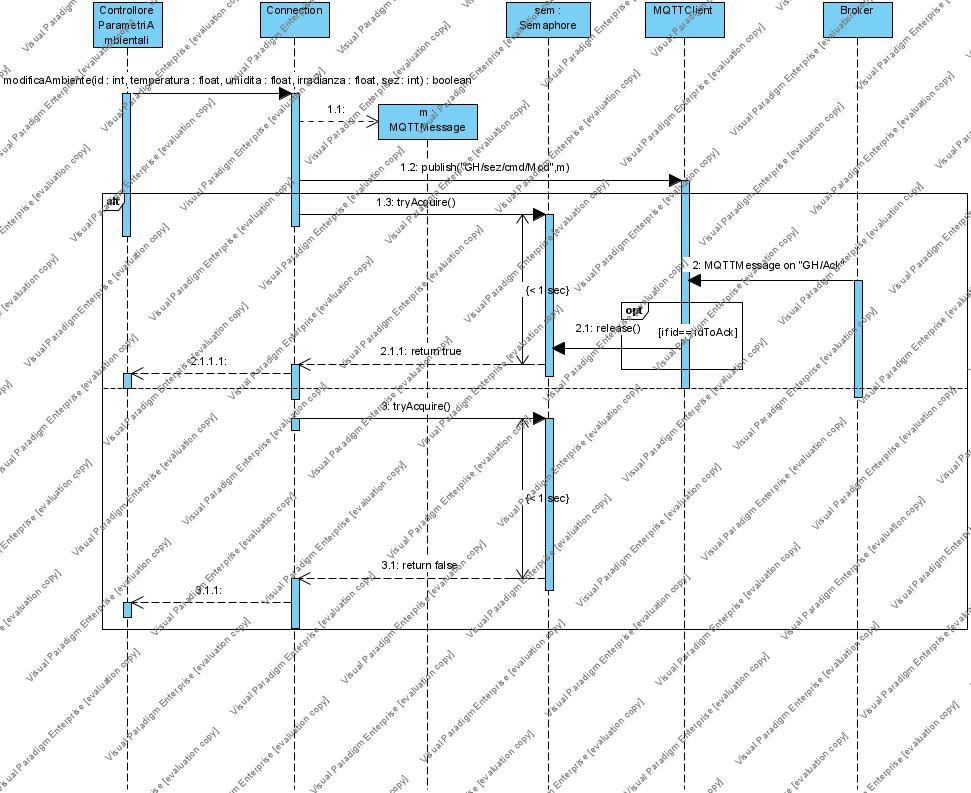
Start-Stop Sequence Diagram

Quando invece è necessario modificare i parametri ambientali target il flusso di controllo prevede che, all’atto della conferma da parte dell’utente sulla modifica, venga inviato un pacchetto col topic di modifica al controllore, il quale prende in consegna il pacchetto e applica correttamente le leggi di attuazione. A quel punto, se non si sono verificati errori, comunica il corretto completamento dell’operazione al wireframe mediante il topic di ack, che a questo punto può salvare a database i parametri inseriti dall’utente. Anche in questo caso, il flusso di comunicazione è descritto nei diagrammi seguenti.

Immagine che contiene mappa, testo

Descrizione generata automaticamente

Modifica parametri ambientali Communication Diagram



*Modifca Ambiente – Richiesta risposta Sequence Diagram*

A questo punto è importante descrivere il funzionamento nel dettaglio delle diverse funzionalità implementate. La differenza principale rispetto ai diagrammi elaborati in fase di analisi sta nel fatto che nei seguenti diagrammi di design vengono specificate le singole chiamate a funzione, le creazioni delle istanze delle classi e i tipi di ritorno, in maniera appunto coerentemente più dettagliata rispetto alla fase di analisi.

## Sequence Diagram di dettaglio – Ricerca coltivazioni

Rispetto alla sua versione di analisi, possiamo vedere chiaramente come il flusso di controllo passa dalla tab di interfaccia dove l’utente inserisce i dati fino alle classi di modello, passando per lo strato di controllo nel quale, come già visto in precedenza, si è preferito ricorrere ad un Facade per abbassare l’accoppiamento tra Layer di controllo ed interfaccia grafica. Notiamo come nel flusso di controllo, al fine di non inviare all’interfaccia grafica classi di modello, la funzione di ricerca coltivazione restituisce delle classi business, nel nostro caso la classe “ColtivazioneBusiness”, la quale racchiude tutti i parametri necessari da mostrare a schermo all’utente. Allo stesso modo, qualora l’utente richiedesse i dettagli della coltivazione, il tipo di ritorno sarebbe di tipo “DettagliBusiness”.

Immagine che contiene testo, mappa, neve, sciando

Descrizione generata automaticamente

Ricerca Coltivazione - Design

## Sequence Diagram di dettaglio – Modifica parametri ambientali target

## Activity Diagram di dettaglio – Controlla parametri ambientali

//sequence diagram di design

Strumenti

Librerie usate eventbus , paho, orm

Framework Hibernate

VM Systems GIthub

1. N.B. Sebbene l’operaio contadino non sia attore univoco di alcun caso d’uso, ci è sembrato comunque corretto includerlo nella gerarchia degli attori in quanto figura professionale riconosciuta dalla serra, a differenza dell’impiegato generico [↑](#footnote-ref-1)