



**UNIVERSITÀ  
DEGLI STUDI  
DI BERGAMO**

UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipartimento di Ingegneria Gestionale, dell'Informazione e della

Produzione

Corso di Laurea in Ingegneria Informatica

Classe: LM-32

SerraSmart

Testing e Verifica del Software

**Studente:**

Gherardi Samuel

Matricola: 1076850

Anno Accademico: 2024/2025



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Asmeta</b>	<b>3</b>
2.1	Obiettivo . . . . .	3
2.2	Requisiti funzionali modellati . . . . .	3
2.3	Domini e struttura del modello ASM . . . . .	4
2.4	Animazione e simulazione . . . . .	4
2.5	Scenari Avalla . . . . .	11
2.6	Proprietà CTL e LTL (Model Checking) . . . . .	11
2.7	Model advisor . . . . .	14
2.8	ATGT - Automatic Test Generation . . . . .	15
2.9	Conclusioni . . . . .	15
<b>3</b>	<b>Implementazione Java con JML</b>	<b>17</b>
3.1	Classe <i>SerraSmartController</i> . . . . .	17
3.2	Specifica JML . . . . .	17
3.3	Verifica dei contratti con OpenJML . . . . .	18
3.4	Considerazioni . . . . .	19
<b>4</b>	<b>Implementazione Java e Testing</b>	<b>21</b>
4.1	Architettura delle classi applicative . . . . .	21
4.1.1	Flusso in modalità AUTOMATICA . . . . .	22
4.1.2	API pubblica essenziale . . . . .	23
4.2	Testing con JUnit5 . . . . .	23
4.3	Esempio di caso di test significativo . . . . .	24

4.4	Considerazioni . . . . .	26
4.5	Mutation Testing con PIT . . . . .	26
4.6	Continuous Integration . . . . .	28
4.7	Analisi Statica del Codice con SonarQube . . . . .	29
4.8	Model-Based Testing: Traduzione da scenario Avalla a JUnit . . . . .	30
4.8.1	Scenario Avalla . . . . .	30
4.8.2	Traduzione in JUnit . . . . .	30
4.8.3	Conclusioni . . . . .	32
4.9	Combinatorial Testing con IPO (Pairwise) . . . . .	32
<b>5</b>	<b>Interfaccia Grafica (UI)</b>	<b>35</b>
5.1	Struttura della UI . . . . .	35
5.2	Test funzionali con Selenium . . . . .	39
5.2.1	Test 1 – Cambio modalità . . . . .	39
5.2.2	Test 2 – Accensione luce manuale . . . . .	40



# Capitolo 1

## Introduzione

Il presente elaborato documenta lo sviluppo e la verifica formale di un sistema software per la gestione intelligente di una serra. Il progetto è stato realizzato con l'obiettivo di applicare tecniche di specifica, modellazione e validazione del comportamento software, secondo il paradigma delle macchine a stati astratte (ASM) e con il supporto del tool ASMETA Studio.

Il sistema *Serra Smart* consente il controllo automatico e manuale di tre categorie di attuatori (luci, ventilatori, irrigatori), sulla base di parametri ambientali monitorati quali temperatura, umidità e luminosità. La modalità automatica consente al sistema di adattare il proprio stato in funzione di soglie configurabili, mentre la modalità manuale permette l'intervento diretto dell'utente sugli attuatori.

Durante lo sviluppo, particolare attenzione è stata dedicata alla simulazione del comportamento ambientale, alla verifica di proprietà logiche tramite tecniche di model checking, e alla generazione automatica di scenari di test. La modellazione è stata realizzata in linguaggio *ASMETAL*, integrando variabili controllate, monitorate e derivate, e successivamente è stata arricchita con test automatici (ATGT) e strumenti di analisi del modello (Model Advisor).

La relazione è suddivisa in sezioni che descrivono i requisiti implementati, il modello ASM sviluppato, la strategia di test adottata e le proprietà formali verificate.

Successivamente, la logica del sistema è stata implementata in *Java*, rispettando una struttura modulare e utilizzando contratti specificati in *JML* (Java Modeling Language) per descrivere in modo formale le precondizioni, postcondizioni e invarianti

delle principali operazioni. Questi contratti sono stati verificati tramite OpenJML per garantire la correttezza formale del codice rispetto al modello.

La fase successiva ha previsto la scrittura di casi di test in *JUnit5*, con l'obiettivo di validare il comportamento funzionale dei componenti della serra sia in modalità automatica che manuale. Per rafforzare l'affidabilità dei test, è stato integrato anche il mutation testing tramite *PIT*, ed è stata configurata una pipeline di Continuous Integration (CI) per automatizzare la fase di testing ad ogni aggiornamento del progetto.

Parallelamente, è stata realizzata una interfaccia grafica (UI) con *Vaadin*, che consente all'utente di monitorare lo stato degli attuatori e agire manualmente o lasciare il controllo in modalità automatica. L'interfaccia è stata testata mediante *Selenium*, con due test principali: uno per il cambio di modalità e uno per l'attivazione manuale degli attuatori.

A seguire, è stata condotta un'analisi statica del codice con *SonarQube*, che ha permesso di identificare code smell e potenziali debiti tecnici, offrendo un'occasione per migliorare la qualità del codice.

Infine, sono stati costruiti casi di test automatici anche secondo il paradigma del model-based testing (trasformando scenari Avalla in test JUnit) e del combinatorial testing usando la tecnica pairwise (IPO algorithm), per ottenere una buona copertura delle combinazioni di soglie senza dover testare l'intero spazio delle configurazioni.

# Capitolo 2

## Asmeta

### 2.1 Obiettivo

Lo scopo di questa prima fase è stato modellare un sistema intelligente per la gestione automatica e manuale della serra, usando il linguaggio *ASMETAL* (Abstract State Machines) tramite l'ambiente di Asmeta. La serra è costituita da:

- 5 luci;
- 2 ventilatori: principale e secondario;
- 3 irrigatori.

### 2.2 Requisiti funzionali modellati

Il sistema modella i seguenti comportamenti:

- attivazione **automatica** di **luci**, **ventilatori** e **irrigatori** in base a soglie ambientali:
  - luminosità;
  - temperatura;
  - umidità.



- Modalità **manuale** in cui l'utente può controllare singolarmente ciascun attuatore;
- commutazione tra modalità AUTOMATICA e MANUALE mediante input monitorato;
- simulazione delle condizioni ambientali tramite *choose* (luminosità, temperatura, umidità);
- condizione di “serra chiusa” se tutti gli attuatori sono spenti.

## 2.3 Domini e struttura del modello ASM

Sono stati definiti i seguenti domini:

- *Luci* e *Irrigator* come subset di interi, *Ventilatori* come enum;
- *StatoLuce*, *LivelloIrrigatore*, *StatoVentilatore* come domini enumerativi;
- *Temperatura*, *Umidita*, *Luminosita* come subset numerici simulati;
- *ModalitaControllo* per distinguere tra modalità automatica e manuale.

Le funzioni ASM sono suddivise in:

- monitored: input esterni (es. *sceltaModalita*, *sogliaTempMax*);
- controlled: stato degli attuatori;
- derived: condizioni logiche (es. *serraChiusa*, *temperaturaTroppoAlta*).

## 2.4 Animazione e simulazione

Per quanto riguarda la gestione automatica della serra, il comportamento dell'ambiente è stato simulato tramite blocchi *choose* all'interno della *main rule*, permettendo di generare in modo casuale

- luminosità;
- temperatura;
- umidità

percepiti all'interno della serra.

In funzione dei valori generati e delle soglie definite, si attivano o disattivano gli attuatori.

In particolare:

1. se la luminosità percepita è inferiore alla soglia minima si accendono tutte le luci;
2. se la luminosità percepita è superiore alla soglia massima si spengono tutte le luci;
3. se la temperatura percepita è inferiore alla soglia minima si spengono tutti i ventilatori e si accendono tutte le luci;
4. se la temperatura percepita è superiore alla soglia massima si accendono tutti i ventilatori;
5. se l'umidità percepita è inferiore alla soglia minima si accendono tutti gli irrigatori;
6. se l'umidità percepita è superiore alla soglia massima si spengono tutti gli irrigatori;

Durante l'animazione sono stati catturati diversi screenshot a supporto della simulazione.

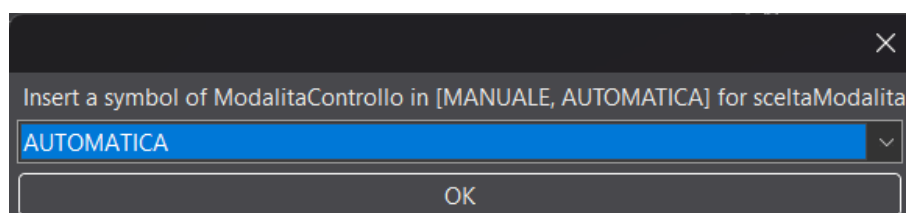
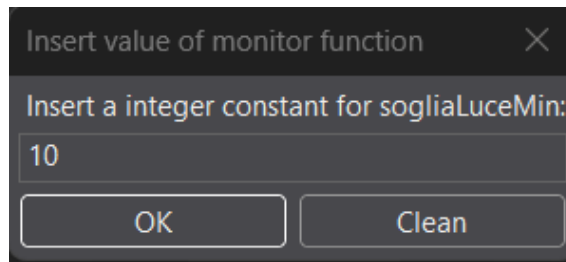


Figura 2.1: Selezione della modalità automatica di gestione della serra



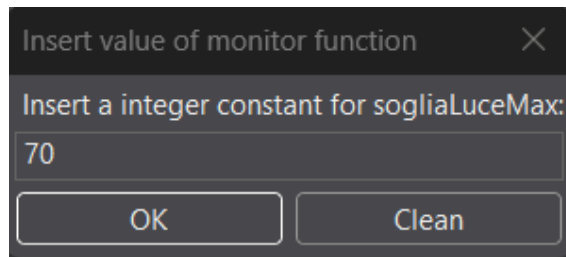
Insert value of monitor function

Insert a integer constant for sogliaLuceMin:

10

OK Clean

Figura 2.2: Definizione della soglia di luminosità minima della serra



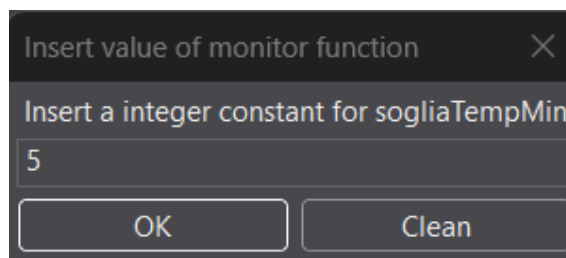
Insert value of monitor function

Insert a integer constant for sogliaLuceMax:

70

OK Clean

Figura 2.3: Definizione della soglia di luminosità massima della serra



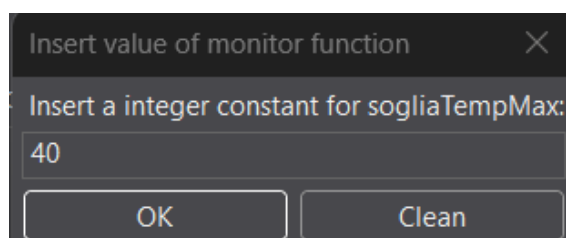
Insert value of monitor function

Insert a integer constant for sogliaTempMin:

5

OK Clean

Figura 2.4: Definizione della soglia di temperatura minima della serra



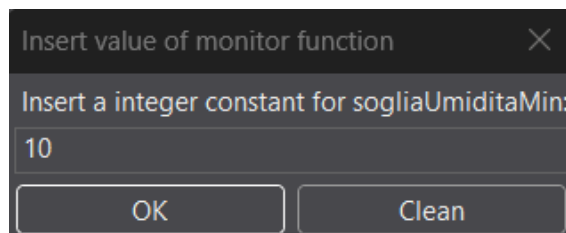
Insert value of monitor function

Insert a integer constant for sogliaTempMax:

40

OK Clean

Figura 2.5: Definizione della soglia di temperatura massima della serra



Insert value of monitor function

Insert a integer constant for sogliaUmiditaMin:

10

OK Clean

Figura 2.6: Definizione della soglia di umidità minima della serra

Insert value of monitor function

Insert a integer constant for sogliaUmiditaMax:

80

OK

Clean

Figura 2.7: Definizione della soglia di umidità massima della serra

	Type	Functions	State 0
<input type="checkbox"/>	M	sceltaModalita	AUTOMATICA
<input type="checkbox"/>	M	sogliaLuceMin	10
<input type="checkbox"/>	M	sogliaLuceMax	70
<input type="checkbox"/>	M	sogliaTempMin	5
<input type="checkbox"/>	M	sogliaTempMax	40
<input type="checkbox"/>	M	sogliaUmiditaMin	10
<input type="checkbox"/>	M	sogliaUmiditaMax	80

Figura 2.8: Stati iniziali definiti

<input type="checkbox"/>	C	statoIrrigatore(2)	0	0
<input type="checkbox"/>	C	statoLuce(4)	OFF	OFF
<input type="checkbox"/>	C	statoIrrigatore(3)	0	0
<input type="checkbox"/>	C	statoLuce(3)	OFF	OFF
<input type="checkbox"/>	C	statoIrrigatore(1)	0	0
<input type="checkbox"/>	C	statoLuce(5)	OFF	OFF
<input type="checkbox"/>	C	statoLuce(2)	OFF	OFF
<input type="checkbox"/>	C	statoLuce(1)	OFF	OFF
<input type="checkbox"/>	C	temperaturaAttuale	Dato che la luminosità rilevata supera la soglia massima, allora tutte le luci vengono spente. L'umidità rilevata è troppo elevata, gli irrigatori restano al livello minimo	26
<input type="checkbox"/>	C	umiditaAttuale		88
<input type="checkbox"/>	C	luminositaAttuale		90

Figura 2.9: Stati iniziali degli attuatori

Type	Functions	State 0	State 1	State 2	State 3	State 4
M	sceltaModalita	AUTO...	AUT...	AUTO...	AUTOMATICA	
M	sogliaLuceMin	45	50	50	65	
M	sogliaLuceMax	90	80	60	80	
M	sogliaTempMin	20	20	20	10	
M	sogliaTempMax	50	45	50	40	
M	sogliaUmiditaMin	20	45	20	10	
M	sogliaUmiditaMax	60	80	50	60	
C	statoIrrigatore(2)	0	0	0	0	0
C	statoIrrigatore(3)	0	0	0	0	0
C	statoIrrigatore(1)	0	0	0	0	0
C	temperaturaAttuale		43	76	76	59
C	umiditaAttuale		86	55	42	42
C	luminositaAttuale		58	83	94	24
C	statoLuce(4)		OFF	OFF	OFF	ON
C	statoLuce(3)		OFF	OFF	OFF	ON
C	statoLuce(5)		OFF	OFF	OFF	ON
C	statoVentilatore(PRINCIPALE)		SPEN...	ACCESO	ACCESO	ACCESO
C	statoLuce(2)		OFF	OFF	OFF	ON
C	statoVentilatore(SECONDARIO)		SPEN...	ACCESO	ACCESO	ACCESO
C	statoLuce(1)		OFF	OFF	OFF	ON

Figura 2.10: Esempio di accensione delle luci

Type	Functions	State 0	State 1	State 2
M	sceltaModalita	AUTOMATICA	AUTOMATICA	
M	sogliaLuceMin	10	30	
M	sogliaLuceMax	70	50	
M	sogliaTempMin	5	15	
M	sogliaTempMax	40	30	
M	sogliaUmiditaMin	10	30	
M	sogliaUmiditaMax	80	70	
C	statoIrrigatore(2)	0	0	0
C	statoLuce(4)	OFF	OFF	OFF
C	statoIrrigatore(3)	0	0	0
C	statoLuce(3)	OFF	OFF	OFF
C	statoIrrigatore(1)	0	0	0
C	statoLuce(5)	OFF	OFF	OFF
C	statoLuce(2)	OFF	OFF	OFF
C	statoLuce(1)	OFF	OFF	OFF
C	temperaturaAttuale		26	97
C	umiditaAttuale		88	65
C	luminositaAttuale		90	93
C	statoVentilatore(PRINCIPALE)		SPENTO	ACCESO
C	statoVentilatore(SECONDARIO)		SPENTO	ACCESO

Figura 2.11: Esempio di accensione dei ventilatori

	Type	Functions	State 0	State 1	State 2
<input checked="" type="checkbox"/>	M	sceltaModalita	AUTO...	AUTO...	
<input checked="" type="checkbox"/>	M	sogliaLuceMin	10	10	
<input checked="" type="checkbox"/>	M	sogliaLuceMax	50	50	
<input checked="" type="checkbox"/>	M	sogliaTempMin	10	10	
<input checked="" type="checkbox"/>	M	sogliaTempMax	50	50	
<input checked="" type="checkbox"/>	M	sogliaUmiditaMin	50	60	
<input checked="" type="checkbox"/>	M	sogliaUmiditaMax	90	80	
<input checked="" type="checkbox"/>	C	statoLuce(4)	OFF	ON	OFF
<input checked="" type="checkbox"/>	C	statoLuce(3)	OFF	ON	OFF
<input checked="" type="checkbox"/>	C	statoLuce(5)	OFF	ON	OFF
<input checked="" type="checkbox"/>	C	statoVentilatore(PRINCIPALE)	SPEN...	SPENTO	ACCESO
<input checked="" type="checkbox"/>	C	statoLuce(2)	OFF	ON	OFF
<input checked="" type="checkbox"/>	C	statoVentilatore(SECONDARIO)	SPEN...	SPENTO	ACCESO
<input checked="" type="checkbox"/>	C	statoLuce(1)	OFF	ON	OFF
<input checked="" type="checkbox"/>	C	temperaturaAttuale		6	87
<input checked="" type="checkbox"/>	C	umiditaAttuale		71	27
<input checked="" type="checkbox"/>	C	luminositaAttuale		93	66
<input checked="" type="checkbox"/>	C	statoIrrigatore(2)	0	100	
<input checked="" type="checkbox"/>	C	statoIrrigatore(3)	0	100	
<input checked="" type="checkbox"/>	C	statoIrrigatore(1)	0	100	

Figura 2.12: Esempio di accensione degli irrigatori

Nella gestione manuale l'utente specifica per ogni attuatore una possibile azione:

- luci: ***ACCENDI\_LUCE*** e ***SPEGNI\_LUCE*** che accende oppure spegne una luce specifica;
- ventilatori: ***ACCENDI\_VENTILATORE*** e ***SPEGNI\_VENTILATORE*** che accende oppure spegne un ventilatore specifico;
- irrigatori: ***APRI\_IRRIGATORE*** e ***CHIUDI\_IRRIGATORE*** che accende al massimo oppure spegne un irrigatore specifico, e ***IMPOSTA\_IRRIGATORE*** che definisce un livello specifico di un irrigatore.

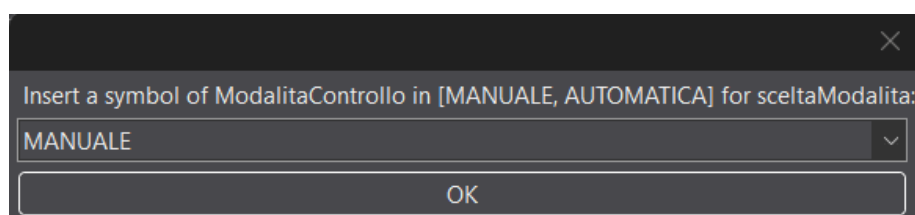


Figura 2.13: Selezione della modalità manuale di gestione della serra

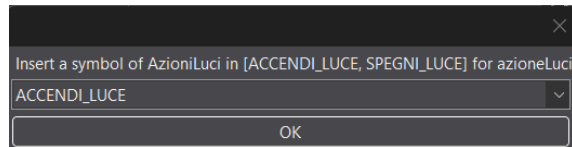


Figura 2.14: Selezione dell'azione di accensione di una luce

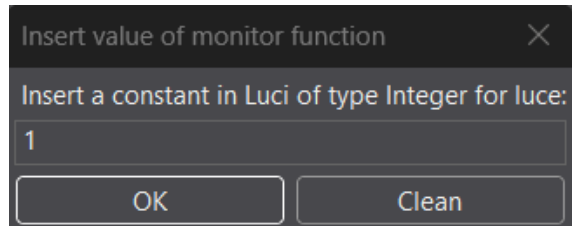


Figura 2.15: Selezione del numero della luce da accendere

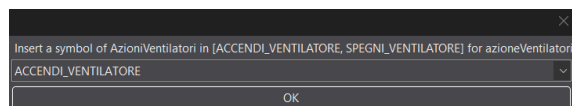


Figura 2.16: Selezione dell'azione di accensione di un ventilatore

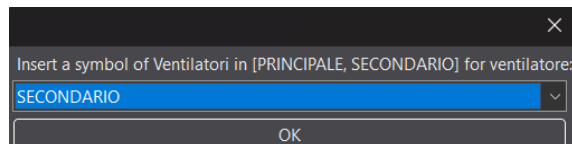


Figura 2.17: Selezione del ventilatore da accendere

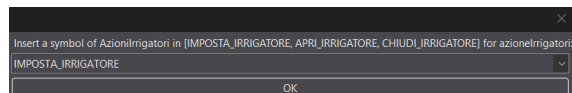


Figura 2.18: Selezione dell'azione di impostazione del livello di un irrigatore

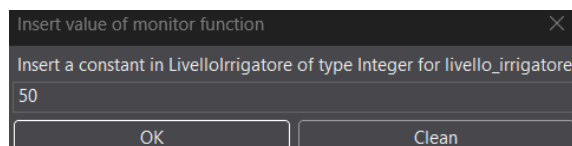


Figura 2.19: Selezione del livello a cui portare l'irrigatore

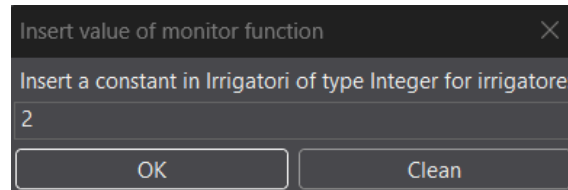


Figura 2.20: Selezione del numero dell'irrigatore da impostare

Type	Functions	State 0	State 1
M	sceltaModalita	MANUALE	
M	azioneLuce	ACCENDI_LUCE	
M	luce	1	
M	azioneVentilatori	ACCENDI_VENTILATORE	
M	ventilatore	SECONDARIO	
M	azioneIrrigatori	IMPOSTA_IRRIGATORE	
M	livello_irrigatore	50	
M	irrigatore	2	
C	temperaturaAttuale		87
C	umiditaAttuale		62
C	luminositaAttuale		75
C	statoIrrigatore(2)		50
C	statoVentilatore(SECONDARIO)		ACCESO
C	statoLuce(1)		ON

Figura 2.21: Stato finale degli attuatori

## 2.5 Scenari Avalla

Sono stati creati scenari di test per verificare manualmente:

- il comportamento automatico completo (luci, ventilatori, irrigatori);
- il comportamento manuale con input dell'utente;
- uno scenario generato automaticamente con *ATGT*.

## 2.6 Proprietà CTL e LTL (Model Checking)

Sono state scritte e verificate alcune proprietà **CTL** (Computation Tree Logic) e **LTL** (Linear Temporal Logic), in modo da:

- verificare correttezza formale;
- trovare errori logici;
- dimostrare proprietà importanti come:



1. *Safety*: “non succede mai qualcosa di sbagliato”;
  2. *Liveness*: “prima o poi succede qualcosa di desiderato”.
- eseguire test esaustivi su comportamenti non coperti dagli scenari.

In particolare sono state definite le seguenti proprietà:

- CTL1 - Liveness: se la temperatura supera la soglia, prima o poi si attiva uno dei due ventilatori;
- CTL2 - Safety: non devono essere accesi contemporaneamente tutti gli attuatori in qualsiasi momento;
- LTL1 - Until: le luci rimangono accese fino a quando la luce non supera la soglia;
- LTL2 - Globale: se la serra è chiusa, nessun attuatore deve essere attivo.

L'esecuzione del model checker di asmeta sul modello originario *SerraSmart* dà dei problemi relativi ai domini di tipo intero, i quali non sono supportati. Per questo motivo si è definito un modello semplificato contenente soltanto domini enumerativi e booleani, *SerraSmartMinimal*, all'interno del quale si modella il funzionamento automatico o manuale della serra, con le seguenti modifiche:

- il dominio delle luci passa da intero a enumerativo;
- il dominio degli irrigatori passa da intero a enumerativo;
- il livello degli irrigatori passa da intero a enumerativo (*MASSIMO*, *MEDIO*, *ZERO*);
- le soglie massime e minime di luminosità, umidità e temperatura non vengono modellizzate;
- il valore di luminosità attuale viene simulato come un enumerativo tra *LALTA*, *LGIUSTA* e *LBASSA*;
- il valore di temperatura attuale viene simulato come un enumerativo tra *TALTA*, *TGIUSTA* e *TBASSA*;

- il valore di umidità attuale viene simulato come un enumerativo tra **UALTA**, **UGIUSTA** e **UBASSA**;
- le luci vengono accese se la luminosità attuale è **LBASSA**, oppure se la temperatura attuale è **TBASSA**;
- i ventilatori vengono accesi se la temperatura attuale è **TALTA**;
- gli irrigatori vengono accesi se l'umidità attuale è **UBASSA**;
- le luci vengono spente se la luminosità attuale è **LALTA**;
- i ventilatori vengono spenti se la temperatura attuale è **TBASSA**;
- gli irrigatori vengono spenti se l'umidità attuale è **UALTA**.

I risultati dell'analisi del model checker sulle proprietà sono riportati in seguito. Si può notare come tutte le proprietà sono state verificate correttamente.

```
Execution of NuSMV code ...
-----
> NuSMV -dynamic -coi -quiet SerraSmartMinimal.smv
-- specification AG ((sceltaModalita = AUTOMATICA & temperaturaAttuale = TALTA) -> EF (statoVentilatore(PRINCIPALE) = ACCESO | statoVentilatore(SECONDARIO) = ACCESO)) is true
model checking (w execution) finished
```

Figura 2.22: Validazione proprietà CTL1

```
> NuSMV -dynamic -coi -quiet SerraSmartMinimal.smv
-- specification IAG ((statoVentilatore(PRINCIPALE) = ACCESO | statoVentilatore(SECONDARIO) = ACCESO) & (((statoLuce(LUCE3) = ON | statoLuce(LUCE2) = ON) | statoLuce(LUCE4) = ON) | statoLuce(LUCE5) = ON) | statoLuce(LUCE1) = ON) & ((statoIrrigatore(IRRIGATORE1) = PASSIVO | statoIrrigatore(IRRIGATORE2) = PASSIVO) | statoIrrigatore(IRRIGATORE3) = PASSIVO))) is true
model checking (w execution) finished
```

Figura 2.23: Validazione proprietà CTL2

```
Execution of NuSMV code ...
-----
> NuSMV -dynamic -coi -quiet SerraSmartMinimal.smv
-- specification G (luminositaAttuale = LBASSA -> (statoLuce(LUCE1) = ON U luminositaAttuale = LBASSA)) is true
model checking (w execution) finished
```

Figura 2.24: Validazione proprietà LTL1

```
Execution of NuSMV code ...
-----
> NuSMV -dynamic -coi -quiet SerraSmartMinimal.smv
-- specification G true is true
model checking (w execution) finished
```

Figura 2.25: Validazione proprietà LTL2

## 2.7 Model advisor

Lo scopo del **model advisor** di asmeta è quello di far eseguire ad asmeta una verifica automatica del modello, per trovare definizioni inutilizzate, variabili mai modificate, regole ridondanti o mai attivate e incoerenze nella semantica ASM.

L'utilizzo del model advisor ha riscontrato le stesse problematiche del model checker, in quanto esso non supporta i domini di tipo intero. Per questo motivo si è eseguito il tool non sul modello originario *SerraSmart*, ma su *SerraSmartMinimal* definita nella sezione precedente 2.6.

I risultati dell'esecuzione sono stati riportati in seguito, l'analisi ha ottenuto un esito positivo.

```
MP1: No inconsistent update is ever performed
NONE

MP3: Choose rule is always/sometimes/never not empty
choose $x in Luminosita with true (when executed) is always not empty.
choose $t in Temperatura with true (when executed) is always not empty.
choose $w in Temperatura with true (when executed) is always not empty.
choose $z in Luminosita with true (when executed) is always not empty.
choose $m in Umidita with true (when executed) is always not empty.
choose $u in Umidita with true (when executed) is always not empty.

MP3: Forall rule is always/sometimes/never not empty
NONE

MP3: Conditional rule eval to true
NONE

MP4: No assignment is always trivial
NONE

MP5: For every domain element e, there exists a location which has value e
NONE

MP6: Every controlled location can take any value in its codomain
NONE

MP7: a location could be removed
NONE

MP7: a controlled location is never updated
NONE

MP7: a controlled location could be static
NONE

model advising finished
```

Figura 2.26: Risultato esecuzione model advisor

## 2.8 ATGT - Automatic Test Generation

È stato creato un file avalla in stile **ATGT** per dimostrare la copertura automatica del sistema.

Il file riproduce un ciclo automatico in cui:

- le condizioni ambientali variano;
- gli attuatori reagiscono in base alle soglie.

Il test è stato animato correttamente e produce lo stato atteso in ogni step.

## 2.9 Conclusioni

La fase di modellazione ASM ha permesso di:

1. descrivere formalmente il sistema *SerraSmart*;
2. verificare la correttezza tramite simulazione e proprietà temporali;
3. generare scenari significativi manuali e automatici.

La base realizzata è stata utilizzata come riferimento per lo sviluppo Java della fase successiva del progetto.



## Capitolo 3

# Implementazione Java con JML

A partire dalla modellazione ASM definita nella fase precedente, si è proceduto con una prima implementazione in linguaggio **Java**. L'obiettivo di questa fase è stato realizzare una componente logica centrale del sistema *SerraSmart*, modellando le soglie ambientali (temperatura, umidità, luminosità) e verificando il comportamento degli attuatori in base a tali valori.

Parte dell'implementazione è stata specificata con l'ausilio di contratti formali tramite il linguaggio **JML** (Java Modeling Language), consentendo una verifica statica delle precondizioni, postcondizioni e invarianti.

### 3.1 Classe *SerraSmartController*

È stata definita la classe *SerraSmartController*, che rappresenta il nucleo di controllo logico della serra. Essa gestisce:

- le soglie configurabili (min/max) per ciascun parametro ambientale;
- la verifica di condizioni critiche (es. temperatura troppo alta);
- le azioni da intraprendere in base al valore rilevato.

### 3.2 Specifica JML

Sulla classe sono stati definiti:

1. **invarianti**: per assicurare la validità dei range delle soglie;
2. **pre-condizioni**: per vincolare l'input dei metodi;
3. **post-condizioni**: per definire il risultato atteso.

Esempio del metodo che restituisce *true* se la temperatura è troppo alta, altrimenti *false* (verificato con OpenJML):

```
//@ requires temperatura <= 100;
//@ ensures \result <==> (temperatura > sogliaTempMax);
public /*@ pure @*/ boolean temperaturaTroppoAlta(int temperatura)
{
    return temperatura > sogliaTempMax;
}
```

Inoltre sono stati specificati anche metodi setter protetti da pre/postcondizioni, come:

```
//@ requires t >= 0 && t <= 100 && t >= sogliaTempMin;
//@ ensures sogliaTempMax == t;
public void setSogliaTempMax(int t) {
    sogliaTempMax = t;
}
```

### 3.3 Verifica dei contratti con OpenJML

I metodi specificati sono stati verificati staticamente con OpenJML. Tutti i metodi hanno dato esito *valid*, incluse le funzioni pure e i metodi setter.

Le seguenti proprietà sono state garantite:

- le soglie restano sempre entro limiti fisici corretti;
- il comportamento logico è deterministico e verificabile;
- le funzioni di controllo sono consistenti con la modellazione ASM.

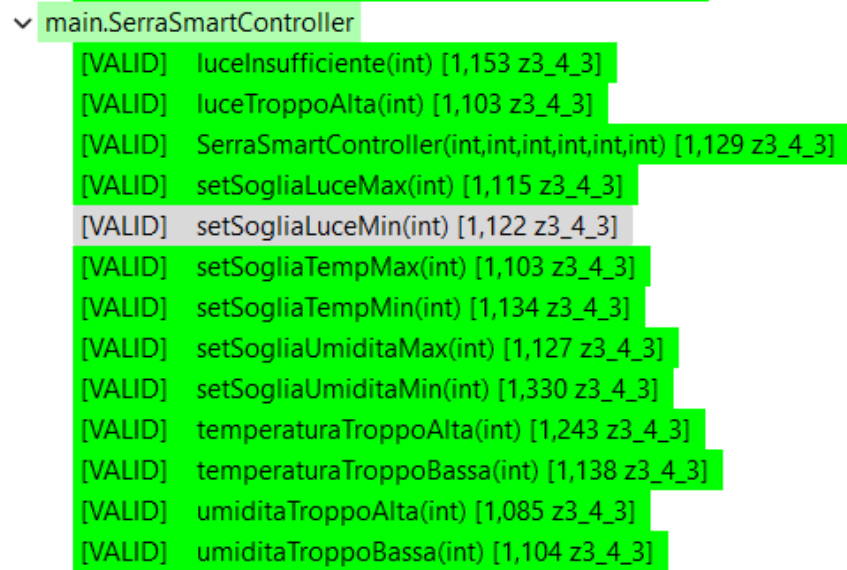


Figura 3.1: Risultato della verifica statica di openJML

## 3.4 Considerazioni

L'uso di JML ha fornito una maggiore confidenza nella correttezza del nucleo decisionale della serra. In particolare:

- gli invarianti impediscono errori di configurazione;
- le post-condizioni assicurano che il risultato sia prevedibile;
- l'integrazione con OpenJML ha reso possibile una verifica automatica del comportamento del sistema, indipendentemente dai test.

La parte successiva dello sviluppo Java estenderà questa logica con test JUnit e componenti applicativi più completi.





# Capitolo 4

## Implementazione Java e Testing

### 4.1 Architettura delle classi applicative

Per estendere la logica definita nel controller JML, sono state introdotte una serie di classi senza contratti formali che mappano in modo diretto gli attuatori della serra:

Classe	Ruolo	Elementi principali
<i>Modalita</i> (enum)	Modalità operative	<i>AUTOMATICA</i> , <i>MANUALE</i>
<i>StatoLuce</i> (enum)	Stato di una luce	<i>ON</i> , <i>OFF</i>
<i>Stato Ventilatore</i> (enum)	Stato di un ventilatore	<i>ACCESO</i> , <i>SPENTO</i>
<i>Ventilatore</i> (enum)	Tipo possibile di ventilatore	<i>PRINCIPALE</i> , <i>SECONDARIO</i>
<i>Centralina Serra</i>	Facciata applicativa che gestisce l'intero impianto	<ul style="list-style-type: none"><li>• Array di luci (5) - HashMap di ventilatori (principale, secondario) - Array di livelli irrigatori (3)</li><li>• Modalità corrente - Logica di attivazione basata su SerraSmartController</li></ul>

Tabella 4.1: Descrizione delle classi, dei loro ruoli ed elementi principali

#### 4.1.1 Flusso in modalità AUTOMATICA

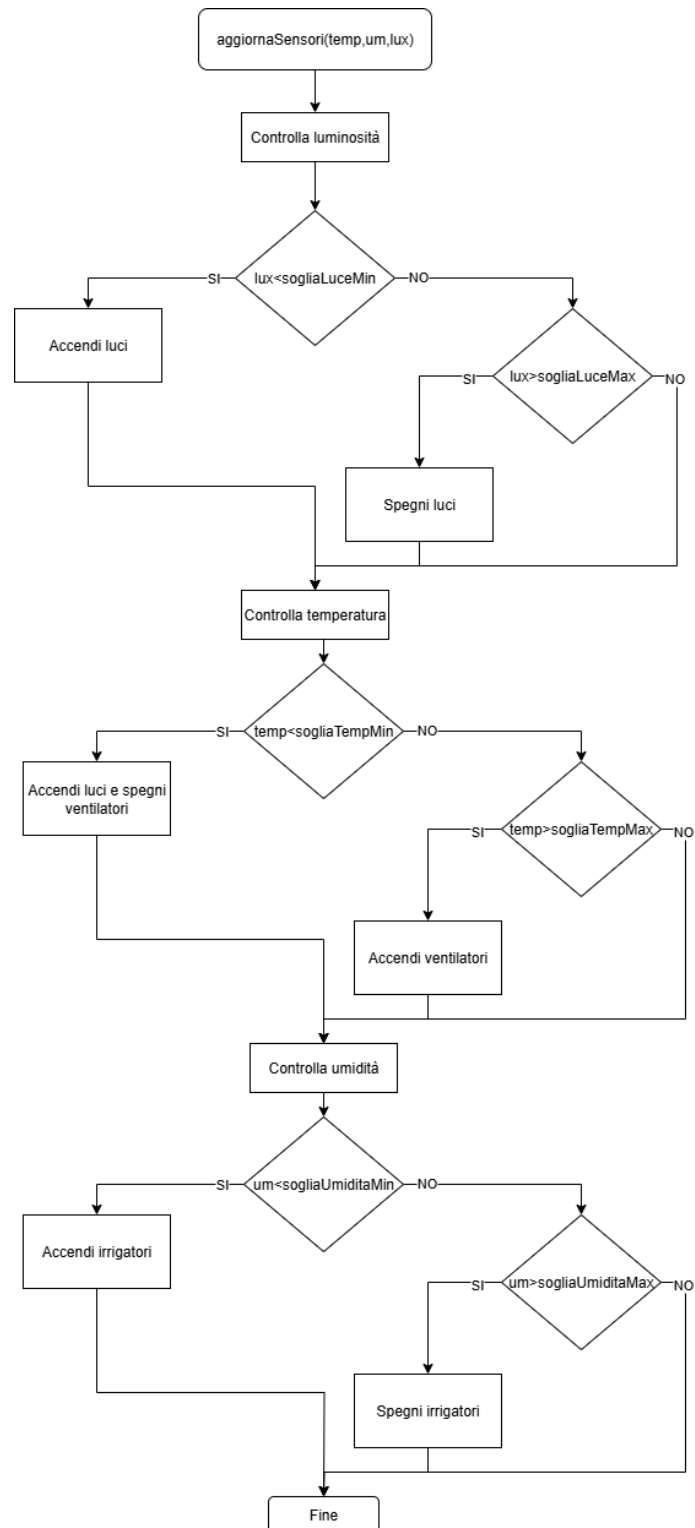


Figura 4.1: Flusso del controllo automatico della serra

### 4.1.2 API pubblica essenziale

```
// cambio modalita
void setModalita(Modalita m);
// aggiornamento sensori (AUTO)
void aggiornaSensori(int t, int u, int lux);
// comandi manuali
void setLuce(int id, StatoLuce s);
void setVentilatore(Ventilatore v, StatoVentilatore s);
void setIrrigatore(int id, int livello);
// getter per i test
StatoLuce getStatoLuce(int id);
StatoVentilatore getStatoVentilatore(Ventilatore v);
int getLivelloIrrigatore(int id);
```

## 4.2 Testing con JUnit5

Per verificare la correttezza funzionale del sistema di controllo della serra, è stata adottata una strategia di testing automatizzato basata su **JUnit 5**, uno dei framework più utilizzati per i test unitari in Java.

L'obiettivo dei test è stato validare:

- il comportamento dei metodi in base ai diversi scenari ambientali;
- la corretta attivazione/disattivazione degli attuatori;
- il rispetto delle modalità operative (AUTOMATICA / MANUALE);
- la gestione dei casi limite (valori errati, indici fuori range, ecc.).

I test sono stati progettati per coprire tutte le diramazioni logiche, compresi i casi di combinazioni di condizioni multiple (copertura MCDC). I casi di test hanno portato ad copertura al 100% delle **istruzioni**, al 100% dei **branch** e al 100% dei **termini**. Per valutare la copertura è stato utilizzato il tool **CodeCover**.

Name	Statement	Branch	Loop	Term
✓ SerraSmartJava	100,0 %	100,0 %	?	100,0 %
✓ main	100,0 %	100,0 %	?	100,0 %
> CentralinaSerra	100,0 %	100,0 %	?	100,0 %
E Modalita	—	—	?	—
> SerraSmartController	100,0 %	—	?	—
E StatoLuce	—	—	?	—
E StatoVentilatore	—	—	?	—
E Ventilatore	—	—	?	—
> test	100,0 %	—	?	100,0 %

Figura 4.2: Copertura ottenuta con i casi di test

Ad esempio, per quanto riguarda la copertura MCDC del metodo *setIrrigatore*, il quale imposta il livello di un particolare irrigatore all'interno della serra, ci si è basati sulla seguente tabella

```
if(i>=0 && i<irrigatori.length && livello>=0 && livello<=100)
```

$i \geq 0$	$i < \text{irrigatori.length}$	$\text{livello} \geq 0$	$\text{livello} \leq 100$	Risultato
T	T	T	T	T
F	T	T	T	F
T	F	T	T	F
T	T	F	T	F
T	T	T	F	F

Tabella 4.2: MCDC *setIrrigatore*

Class: CentralinaSerra		Condition: (((i>=0 AND i<irrigatori.length) AND livello>=0) AND livello<=100)											
(	(	i>=0	AND	i<irrigatori.length	)	AND	livello>=0	)	AND	livello<=100	)	Result	Test Cases (Number of Executions)
		T	T	T		F	F		F	x		0	test.CentralinaSerraTest.testMCDC (1) Coverage: 12,0
		T	T	T		T	T		F	F		0	test.CentralinaSerraTest.testMCDC (1) Coverage: 12,0
		T	T	T		T	T		T	T		1	test.CentralinaSerraTest.testMCDC (1), test.CentralinaSerraTest.testManualeModificaStato (1) Coverage: 50,0
		F	F	x		F	x		F	x		0	test.CentralinaSerraTest.testMCDC (1), test.CentralinaSerraTest.testModificheErrate (1) Coverage: 12,0
		T	F	F		F	x		F	x		0	test.CentralinaSerraTest.testMCDC (1) Coverage: 12,0

Figura 4.3: MDCD di *setIrrigatore* CodeCover

### 4.3 Esempio di caso di test significativo

Uno dei test più rappresentativi dell'intero sistema è quello in cui si verifica il comportamento della centralina in modalità automatica di fronte a un insieme di condizioni ambientali critiche.

### Scenario simulato:

Viene richiamato il metodo *aggiornaSensori(40, 90, 100)* con i seguenti valori:

- temperatura = 40 °C;
- umidità = 90%;
- luminosità = 100 lux.

Confrontando questi valori con le soglie predefinite:

- sogliaTempMax = 30 → temperatura troppo alta;
- sogliaUmiditaMax = 70 → umidità troppo alta;
- sogliaLuceMin = 200 → luminosità troppo bassa.

### Azioni attese dal sistema:

In base alla logica implementata nella centralina:

- le luci devono accendersi, poiché la luminosità è insufficiente;
- i ventilatori devono attivarsi, per abbassare la temperatura;
- gli irrigatori devono spegnersi, per ridurre l'umidità eccessiva.

### Verifiche effettuate nel test:

I test *testLuceInsufficienteAccendeLuci()*, *testTemperaturaTroppoAltaAccendeVentilatori()* e *testUmiditaTroppoAltaSpegneIrrigatori()* verificano che:

```
assertEquals(StatoLuce.ON, centralina.getStatoLuce(0));
assertEquals(StatoVentilatore.ACCESO, centralina.
    getStatoVentilatore(Ventilatore.PRINCIPALE));
assertEquals(0, centralina.getLivelloIrrigatore(0));
```

Questo caso mette in evidenza come il sistema, a partire da un singolo aggiornamento ambientale, interagisca con più attuatori in parallelo applicando regole diverse in funzione dei singoli sensori, garantendo così una reazione completa e coerente.

## 4.4 Considerazioni

L'implementazione della logica della serra in Java ha rappresentato la naturale evoluzione del modello formale realizzato in ASMETA. Le classi definite, in particolare *CentralinaSerra* e *SerraSmartController*, riflettono in modo diretto le regole e le condizioni descritte nel modello ASM, consentendo una transizione fluida tra specifica e codice eseguibile.

Particolare attenzione è stata dedicata alla distinzione tra modalità automatica e modalità manuale, replicando fedelmente le scelte comportamentali previste nel sistema reattivo. La centralina, infatti, decide in autonomia come agire sugli attuatori solo se la modalità automatica è attiva, altrimenti attende esplicitamente comandi manuali.

Per quanto riguarda il testing, l'adozione di JUnit 5 ha permesso una copertura completa delle funzionalità più critiche del sistema, sia nei casi standard sia nei casi limite. L'integrazione tra le classi applicative e i contratti JML definiti in precedenza ha inoltre permesso di mantenere una chiara separazione tra la logica di business e i vincoli formali, rafforzando l'affidabilità dell'intero sistema.

Nel complesso, questa fase ha confermato l'efficacia di una progettazione guidata da modelli (Model-Driven), in cui la formalizzazione iniziale e i test automatici contribuiscono congiuntamente a garantire correttezza, manutenzione e tracciabilità del software sviluppato.

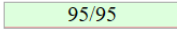
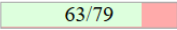
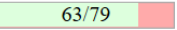
## 4.5 Mutation Testing con PIT

Per valutare in modo più approfondito la capacità del sistema di rilevare anomalie nel comportamento, è stato eseguito un test di mutation testing utilizzando lo strumento **PIT Mutation Testing** integrato in Eclipse.

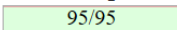
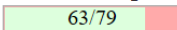
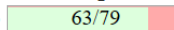
PIT ha generato 79 mutanti, introducendo piccole variazioni nei metodi della classe *CentralinaSerra* e della classe *SerraSmartController*. Di questi, 63 mutanti sono stati correttamente rilevati e uccisi dai test, mentre 16 sono sopravvissuti. Questo si traduce in una mutation coverage dell'80%, considerata un'ottima soglia nei progetti reali. La figura seguente riporta il riepilogo del test eseguito:

# Pit Test Coverage Report

## Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	100%  95/95	80%  63/79	80%  63/79

## Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
<a href="#">main</a>	2	100%  95/95	80%  63/79	80%  63/79

Report generated by [PIT](#) 1.6.8

Figura 4.4: Riepilogo mutation testing

La maggior parte delle mutazioni sopravvissute si riferisce a:

- condizioni booleane non rilevanti ai fini del comportamento;
- rami che, per costruzione del codice, non alterano l'output;
- mutanti equivalenti, ossia che non modificano il comportamento osservabile del sistema.

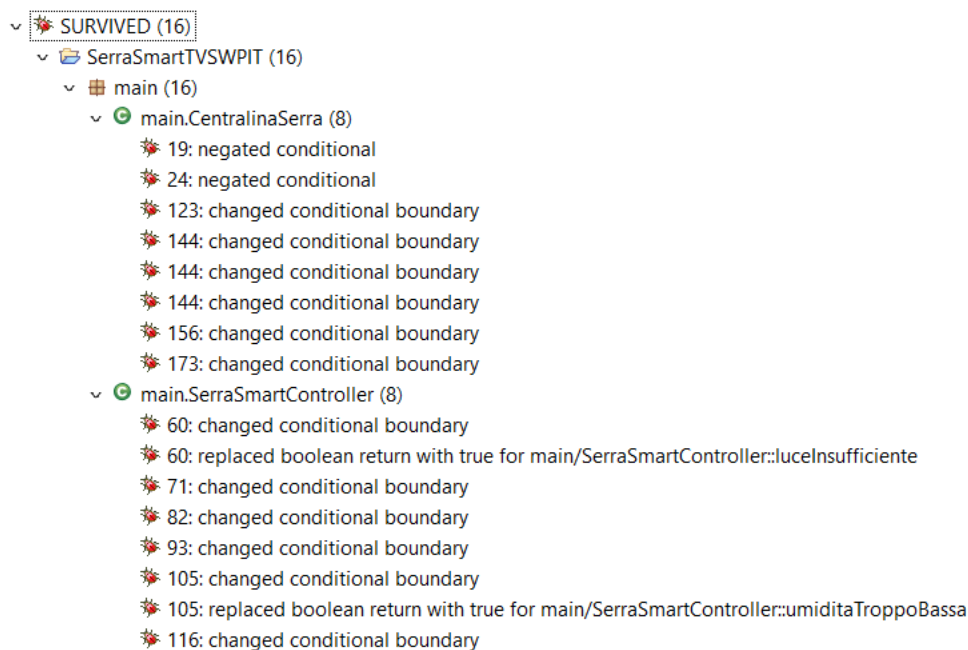


Figura 4.5: Mutazioni sopravvissute



In sintesi, il test ha confermato la solidità della test suite, già validata con il coverage strutturale (branch/MCDC), e ha fornito ulteriori garanzie sulla robustezza logica e comportamentale della centralina automatizzata.

## 4.6 Continuous Integration

Per migliorare la qualità del codice e automatizzare il processo di build e test, ho introdotto una pipeline di **Continuous Integration** (CI) utilizzando **GitHub Actions**.

La pipeline viene attivata automaticamente ad ogni push o pull request sul repository. Il workflow esegue i seguenti passi:

- checkout del codice sorgente;
- setup dell'ambiente Java (versione 23);
- download della libreria JUnit Console per l'esecuzione dei test;
- compilazione del codice sorgente e dei test;
- esecuzione automatica dei test unitari con report dei risultati.

Di seguito è riportato uno screenshot della GitHub Actions che mostra l'esito positivo dell'ultimo run della pipeline. Come si può vedere, la compilazione e i test sono stati eseguiti con successo, garantendo un controllo continuo sulla qualità del codice.

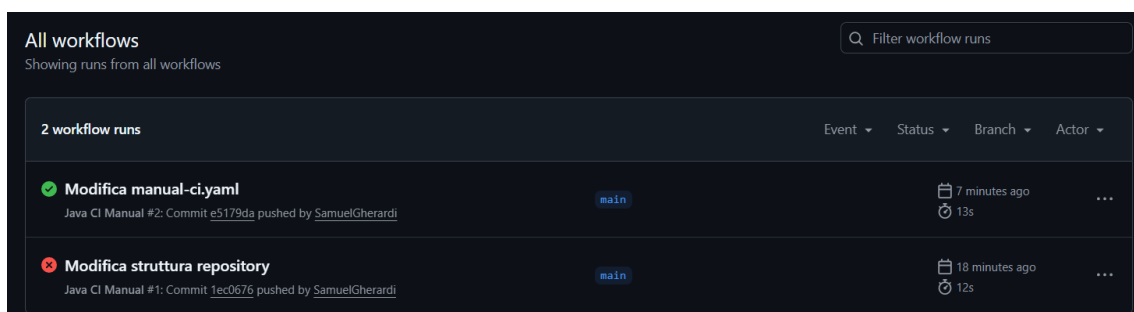


Figura 4.6: Continuous Integration con GitHub Actions

## 4.7 Analisi Statica del Codice con SonarQube

Per valutare la qualità del codice Java dell'applicazione *SerraSmart*, è stata condotta un'analisi statica tramite **SonarQube**, una piattaforma avanzata per il controllo continuo del codice.

L'analisi ha evidenziato:

- code smell minori (24 segnalazioni in totale);
- nessun bug critico né vulnerabilità di sicurezza;
- Presenza di blocchi di codice commentato da rimuovere (dovuti a openJML);
- variabili non conformi al naming convention ( $\hat{[a-z][a-zA-Z0-9]*\$}$ );
- suggerimenti di miglioramento come l'uso di EnumMap e del diamond operator ( $<>$ ) nei costruttori.

24 items			
Resource	Date	Description	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+1 location]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+11 locations]	
SerraSmartCor	6 days ago	⚠ This block of commented-out lines of code should be removed. [+8 locations]	
CentralinaSerr.	4 days ago	⚠ This block of commented-out lines of code should be removed.	
CentralinaSerr.	4 days ago	⚠ This block of commented-out lines of code should be removed.	
CentralinaSerr.	4 days ago	⚠ This block of commented-out lines of code should be removed.	
CentralinaSerr.	5 days ago	⚠ Convert this Map to an EnumMap.	
CentralinaSerr.	5 days ago	⚠ Merge this if statement with the enclosing one. [+1 location]	
CentralinaSerr.	5 days ago	⚠ Merge this if statement with the enclosing one. [+1 location]	
CentralinaSerr.	5 days ago	⚠ Rename this local variable to match the regular expression $^{[a-z][a-zA-Z0-9]*\$}$ .	
CentralinaSerr.	5 days ago	⚠ Rename this local variable to match the regular expression $^{[a-z][a-zA-Z0-9]*\$}$ .	
CentralinaSerr.	5 days ago	⚠ Replace the type specification in this constructor call with the diamond operator ( $<>$ ).	

Figura 4.7: Output dell'analisi statica effettuata da SonarQube

In generale non sono emerse criticità gravi nel codice, e le segnalazioni possono essere facilmente risolte per migliorare:

- la leggibilità;
- la manutenibilità;
- la conformità agli standard di codifica.

## 4.8 Model-Based Testing: Traduzione da scenario

### Avalla a JUnit

In questa sezione viene mostrata l'applicazione del **Model-Based Testing** attraverso la traduzione di uno scenario definito in Avalla in un test automatico scritto in JUnit, al fine di verificare che l'implementazione in Java rispetti il comportamento previsto dal modello ASM.

#### 4.8.1 Scenario Avalla

Lo scenario utilizzato verifica il comportamento della serra in modalità automatica, forzando una sequenza di valori sensoriali e controllando lo stato degli attuatori. In particolare, vengono valutati:

- l'accensione delle luci in condizioni di bassa luminosità;
- l'attivazione dei ventilatori per temperatura elevata;
- l'attivazione completa degli irrigatori per bassa umidità;
- i successivi cambiamenti di stato a fronte di nuove condizioni ambientali.

#### 4.8.2 Traduzione in JUnit

Lo scenario è stato tradotto nel test *AvallaTest()* riportato di seguito:

```
@Test
public void AvallaTest() {
    SerraSmartController controller = new SerraSmartController(10,
        50, 5, 30, 10, 50);
    CentralinaSerra centralina = new CentralinaSerra(5, 3,
        controller, Modalita.AUTOMATICA);

    // Verifica stato iniziale
    for (int i = 0; i < 5; i++) assertEquals(StatoLuce.OFF,
        centralina.getStatoLuce(i));
    assertEquals(StatoVentilatore.SPENTO, centralina.
        getStatoVentilatore(Ventilatore.PRINCIPALE));
```

```

assertEquals(StatoVentilatore.SPENTO, centralina.
    getStatoVentilatore(Ventilatore.SECONDARIO));
for (int i = 0; i < 3; i++) assertEquals(0, centralina.
    getLivelloIrrigatore(i));

// Step 1: condizioni critiche
centralina.aggiornaSensori(40, 9, 8);
for (int i = 0; i < 5; i++) assertEquals(StatoLuce.ON,
    centralina.getStatoLuce(i));
assertEquals(StatoVentilatore.ACCESO, centralina.
    getStatoVentilatore(Ventilatore.PRINCIPALE));
assertEquals(StatoVentilatore.ACCESO, centralina.
    getStatoVentilatore(Ventilatore.SECONDARIO));
for (int i = 0; i < 3; i++) assertEquals(100, centralina.
    getLivelloIrrigatore(i));

// Step 2: temperatura fuori soglia
centralina.aggiornaSensori(40, 65, 60);
for (int i = 0; i < 5; i++) assertEquals(StatoLuce.OFF,
    centralina.getStatoLuce(i));
assertEquals(StatoVentilatore.ACCESO, centralina.
    getStatoVentilatore(Ventilatore.PRINCIPALE));
assertEquals(StatoVentilatore.ACCESO, centralina.
    getStatoVentilatore(Ventilatore.SECONDARIO));
for (int i = 0; i < 3; i++) assertEquals(0, centralina.
    getLivelloIrrigatore(i));

// Step 3: temperatura bassa
centralina.aggiornaSensori(3, 65, 60);
for (int i = 0; i < 5; i++) assertEquals(StatoLuce.ON,
    centralina.getStatoLuce(i));
assertEquals(StatoVentilatore.SPENTO, centralina.
    getStatoVentilatore(Ventilatore.PRINCIPALE));
assertEquals(StatoVentilatore.SPENTO, centralina.
    getStatoVentilatore(Ventilatore.SECONDARIO));
for (int i = 0; i < 3; i++) assertEquals(0, centralina.
    getLivelloIrrigatore(i));

```

```
}
```

### 4.8.3 Conclusioni

Questo test garantisce una copertura **coerente** con lo scenario Avalla, dimostrando che l'implementazione Java è conforme alle specifiche modellate in ASM. Questo approccio rafforza la coerenza tra modello e codice e si rivela utile per la validazione del comportamento complessivo della serra.

## 4.9 Combinatorial Testing con IPO (Pairwise)

In questa fase si è applicato il **combinatorial testing** utilizzando l'algoritmo **IPO** (In-Parameter-Order). Sono stati definiti 3 parametri ambientali (luminosità, temperatura, umidità), ognuno con 3 livelli:

- luminosità: [5, 30, 80];
- temperatura: [2, 20, 40];
- umidità: [5, 40, 90].

Invece di testare tutte le 27 combinazioni, è stato utilizzato l'algoritmo IPO per coprire tutte le coppie significative (pairwise), ottenendo 9 casi di test.

Luminosità	Temperatura	Umidità
5	2	5
5	20	40
5	40	90
30	20	90
30	40	5
30	2	40
80	40	40
80	2	90
80	20	5

Tabella 4.3: Casi di test mediante combinatorial testing

Per ogni combinazione, è stato scritto un test parametrizzato JUnit che forza i valori dei sensori e verifica lo stato atteso degli attuatori:

- luci accese se luminosità  $< 10$ ;
- ventilatori accesi se temperatura  $> 30$ ;
- irrigatori attivati al 100% se umidità  $< 10$ .

Questa tecnica ha permesso di coprire efficacemente interazioni rilevanti tra i parametri con un numero contenuto di test, individuando rapidamente configurazioni potenzialmente critiche o anomale.



# Capitolo 5

## Interfaccia Grafica (UI)

L'interfaccia utente è stata sviluppata utilizzando *Vaadin 24*, un framework Java per applicazioni web che consente di costruire componenti lato server in modo dichiarativo, mantenendo un design reattivo e moderno.


### 5.1 Struttura della UI

L'interfaccia principale dell'applicazione si presenta come una dashboard suddivisa in diverse sezioni, in base alla modalità operativa della serra:

- **barra superiore di stato:** mostra informazioni sullo stato di attuatori e sensori (luci, ventilatori, irrigatori), aggiornate dinamicamente. Lo stato è reso visivamente con badge colorati:
  - **verde:** luce attiva, irrigatore completamente aperto;
  - **grigio:** attuatore inattivo;
  - **rosso:** ventilatore acceso;
  - **blu:** irrigatore aperto al di sotto del 50%.
- **pulsante “Cambia modalità”:** consente il passaggio tra modalità automatica e modalità manuale;
- **sezione di configurazione soglie** (visibile solo in modalità automatica):



- permette all’utente di impostare le soglie minime e massime per temperatura, umidità e luminosità;
- i campi sono distribuiti in tre colonne, una per ciascun parametro ambientale;
- **sezione di controllo manuale** (visibile solo in modalità manuale):
  - contiene comandi per accendere o spegnere singole luci;
  - comandi per attivare o disattivare ventilatori;
  - comandi per impostare il livello di apertura degli irrigatori;
  - anche questa sezione è distribuita in colonne per maggiore chiarezza.
- **Pulsante “Simula dati”**: genera valori casuali per temperatura, umidità e luce. I dati simulati vengono mostrati a schermo in tempo reale.


Serra Smart UI

Modalità attuale: AUTOMATICA

**LUCI:**
L0: OFF
L1: OFF
L2: OFF
L3: OFF
L4: OFF

**VENTILATORI:**
PRINCIPALE: SPENTO
SECONDARIO: SPENTO

**IRRIGATORI:**
I0: 0%
I1: 0%
I2: 0%

Cambia modalità


Simula dati casuali

### Configurazione Soglie

Soglie Luce	Soglie Temperatura	Soglie Umidità
Soglia luce min	Soglia temperatura min	Soglia umidità min
200	15	30
Soglia luce max	Soglia temperatura max	Soglia umidità max
800	30	70

Aggiorna soglie

Figura 5.1: UI della gestione automatica della serra


Serra Smart UI

Modalità attuale: MANUALE

**LUCI:**
L0: OFF
L1: OFF
L2: OFF
L3: OFF
L4: OFF

**VENTILATORI:**
PRINCIPALE: SPENTO
SECONDARIO: SPENTO

**IRRIGATORI:**
I0: 0%
I1: 0%
I2: 0%

Cambia modalità

Simula dati casuali

### Controllo Manuale

Luci	Ventilatori	Irrigatori
Indice luce (0-4)	Ventilatore	Indice irrigatore (0-2)
Accendi luce	Accendi ventilatore	Livello apertura (0-100)
Spegni luce	Spegni ventilatore	
		Imposta irrigatore

Figura 5.2: UI della gestione manuale della serra

Modalità attuale: AUTOMATICA

LUCI: L0: OFF L1: OFF L2: OFF L3: OFF L4: OFF

VENTILATORI: PRINCIPALE: ACCESO SECONDARIO: ACCESO

IRRIGATORI: I0: 100% I1: 100% I2: 100%

Cambia modalità

Simula dati casuali

Dati simulati: 32°C | 9% | 803lx

### Configurazione Soglie

Soglie Luce	Soglie Temperatura	Soglie Umidità
Soglia luce min	Soglia temperatura min	Soglia umidità min
200	15	30
Soglia luce max	Soglia temperatura max	Soglia umidità max
800	30	70

Aggiorna soglie

Figura 5.3: Esempio di simulazione dati in modalità automatica

Modalità attuale: AUTOMATICA

LUCI: L0: OFF L1: OFF L2: OFF L3: OFF L4: OFF

VENTILATORI: PRINCIPALE: ACCESO SECONDARIO: ACCESO

IRRIGATORI: I0: 100% I1: 100% I2: 100%

Cambia modalità

Simula dati casuali

✔ Soglie aggiornate correttamente.

### Configurazione Soglie

Soglie Luce	Soglie Temperatura	Soglie Umidità
Soglia luce min	Soglia temperatura min	Soglia umidità min
200	15	30
Soglia luce max	Soglia temperatura max	Soglia umidità max
800	50	70

Aggiorna soglie

Figura 5.4: Esempio di aggiornamento soglie in modalità automatica

Modalità attuale: MANUALE

LUCI: L0: OFF L1: OFF L2: ON L3: OFF L4: OFF

VENTILATORI: PRINCIPALE: ACCESO SECONDARIO: SPENTO

IRRIGATORI: I0: 100% I1: 45% I2: 100%

Cambia modalità

Simula dati casuali

Dati simulati: 9°C | 86% | 356lx

### Controllo Manuale

Luci	Ventilatori	Irrigatori
Indice luce (0-4)	Ventilatore	Indice irrigatore (0-2)
2	SECONDARIO	1
Accendi luce	Accendi ventilatore	Livello apertura (0-100)
Spegni luce	Spegni ventilatore	45
		Imposta irrigatore

Figura 5.5: Esempio di accensione/spegnimento di attuatori in modalità manuale

## 5.2 Test funzionali con Selenium

Per verificare il corretto funzionamento dell'interfaccia utente realizzata con Vaadin, sono stati sviluppati due test automatici utilizzando **Selenium WebDriver**. I test sono stati eseguiti tramite Eclipse, utilizzando il browser Google Chrome e il relativo ChromeDriver.

### 5.2.1 Test 1 – Cambio modalità

**Obiettivo:** simulare il comportamento dell'utente che, una volta effettuato il login, preme il pulsante “Cambia modalità” per passare dalla modalità automatica a quella manuale.

**Fasi:**

1. avvio del browser e apertura della pagina *http://localhost:8080*;
2. compilazione dei campi di login;
3. clic sul pulsante “Cambia modalità”;

4. verifica che l'etichetta relativa alla modalità cambi in "Modalità attuale: MANUALE".

**Esito:** Test superato con successo.

```
🔧 Avvio test Selenium  
☑ Login e cambio modalità riusciti!  
🔧 Chiudo browser
```

Figura 5.6: Esito del primo test Selenium

### 5.2.2 Test 2 – Accensione luce manuale

**Obiettivo:** verificare che, in modalità manuale, l'utente possa accendere una singola luce della serra.

**Fasi:**

1. login e cambio modalità in MANUALE;
2. inserimento dell'indice della luce (es. 0);
3. clic sul pulsante "Accendi luce";
4. verifica che lo stato della luce L0 diventi "ON" nella barra informativa.

**Esito:** Test superato con successo.

```
🔧 Test: accensione luce manuale  
☑ Luce 0 accesa manualmente con successo  
🔧 Chiudo browser
```

Figura 5.7: Esito del secondo test Selenium