

Progetto di Testing e Verifica del Software

Introduzione al progetto

Si è sviluppato un progetto Java per il controllo di una **smart home**. La feature principale è poter controllare la chiusura (a chiave) delle porte, il livello di chiusura (in percentuale) delle tapparelle e lo stato delle luci (accese, spente e soffuse). I requisiti riguardanti questa parte sono i seguenti:

- La smart home è dotata di 5 luci, 3 tapparelle (generalizzati in Java), 3 porte (PRINCIPALE, RETRO e GARAGE) e un sistema antifurto.
- Nell'ASMETA, con probabilità 15% si modella il tentativo di furto, altrimenti si modella un'azione dell'utente. In Java il tentativo di furto non è casuale ma corrisponde alla chiamata di un metodo.
- Quando scatta l'allarme, se l'antifurto è inserito, si chiudono porte e tapparelle e si accendono le luci.
- Quando si attiva l'antifurto, la porta principale si chiude e l'antifurto non può essere disattivato se la porta principale è aperta.
- È possibile aprire e chiudere l'intera casa: le azioni totali sulla casa aprono (o chiudono) tutte le porte e tutte le tapparelle. Se si chiude tutto, viene attivato anche l'antifurto.
- È possibile accendere, spegnere o rendere soffusa una luce a scelta.
- È possibile chiudere e aprire completamente tutte le tapparelle e impostare la percentuale di chiusura di una a scelta.
- È possibile aprire o chiudere (a chiave) una porta a scelta. Quando apro la porta principale, se nessuna luce è accesa, tutte le tapparelle chiuse più del 50%, si aprono al 50%. La porta si può chiudere solo se l'antifurto è attivo.

Per questa parte è stato sviluppato un modello in ASMETA, implementato successivamente in Java.

Inoltre, sempre nell'ottica della smart home, è stata implementata la logica per il controllo e la lettura dei dati di una **stazione meteorologica** e per il controllo di una **vasca idromassaggio**. Le specifiche sono le seguenti:

- Per la stazione meteorologica: dati in input temperatura esterna (in °C) e umidità relativa (%) deve essere calcolato il punto di rugiada (*dew point*) e dati in input temperatura esterna (in °C), UV index (intero positivo), tasso di pioggia (mm/hr) e vento (km/h) deve segnalare se sussiste una condizione d'allerta o no. Inoltre, la stazione meteorologica mantiene i valori di temperatura e umidità relativa interni degli ultimi sette giorni e restituisce il massimo valore di temperatura e il valor medio di umidità relativa.
- Per la vasca idromassaggio: è possibile controllare il livello di riempimento, svuotando o riempiendo completamente la vasca o svuotarla o riempirla di un ulteriore 25%.

Anche per la vasca idromassaggio sono stati implementati sia il modello (ASM) che codice Java.

Il sistema si divide in tre progetti, un progetto Java gestito da Maven e due progetti ASMETA contenenti i modelli. La struttura generale è riassunta di seguito:

- Progetto SmartHome
 Contiene i package `smart_home` e `smart_home_additional_elements`, il primo contiene a sua volta la classe `SmartHome.java`, tre enumerativi utilizzati al suo interno e le classi di test. Il secondo package contiene le classi `StazioneMeteo.java` e `VascaIdromassaggio.java` e le relative classi di test.

- Progetto SmartHomeModel e VascaIdromassaggioModel
Contengono rispettivamente la macchina smarthome.asm, le librerie necessarie al suo funzionamento, uno scenario avalla e la macchina vascaidro.asm, le librerie e una serie di scenari generati automaticamente.

L'obiettivo del progetto è di utilizzare alcune tecniche di testing e verifica del codice e testing e verifica basati sui modelli sugli artefatti software descritti in precedenza. L'applicazione di tali tecniche è descritta di seguito. Si fa notare che l'IDE utilizzato è Eclipse in aggiunta a diversi tool, framework e plug-in compatibili.

Code Testing

Su tutto il codice Java prodotto è stato fatto testing di unità sfruttando JUnit (versione 4). L'obiettivo principale, oltre a far passare tutti i test, era ottenere una **copertura degli statement e dei branch** del 100% sull'intero progetto. Per verificare che tale copertura è stata raggiunta si è utilizzato il tool CodeCover.

Name	Statement	Branch	Loop	Term	?-Operator	Synchronized
SmartHome	100,0 %	100,0 %	?	97,1 %	?	?
smart_home	100,0 %	100,0 %	?	94,7 %	?	?
Porta	—	—	?	—	?	?
SmartHome	100,0 %	100,0 %	?	94,7 %	?	?
StatoLuce	—	—	?	—	?	?
StatoPorta	—	—	?	—	?	?
smart_home_additional	100,0 %	100,0 %	?	100,0 %	?	?
StazioneMeteo	100,0 %	100,0 %	?	100,0 %	?	?
Vascaldromassaggio	100,0 %	100,0 %	?	100,0 %	?	?

Si può notare che anche la copertura dei term del secondo package è al 100%, per ottenere ciò è stato applicato il **criterio MCDC** alla classe StazioneMeteo (mentre per Vascaldromassaggio si è sfruttato il model-based testing, come descritto in seguito). Di seguito viene riportata la tabella definita per ottenere tale copertura:

`T <= -10 || T >= 30 || UVI > 5 || (rainRate > 10 && wind > 50) || (T<=0 && rainRate > 0)`










































T<=10	T>=30	UVI>5	rainRate>10	Wind>50	T<=0	rainRate>0	Decisione
T	-	-	-	-	-	-	T
F	T	-	-	-	-	-	T
F	F	T	-	-	-	-	T
F	F	F	T	T	-	-	T
F	F	F	F	-	T	T	T
F	F	F	F	-	F	-	F
F	F	F	T	F	F	-	F
F	F	F	F	-	T	F	F

*dove c'è una barra – il valore non viene valutato a causa della lazy evaluation

Inoltre, per testare il metodo che calcola il dew point della classe StazioneMeteo è stata utilizzata la funzione dei **test parametrici** di JUnit visto l'elevata quantità di input utilizzati nel test (25 casi di test ottenuti con input domain modeling e combinatorial testing ACoC e che hanno portato al 100% di copertura, come descritto più avanti).

Nel corso dello sviluppo del progetto è stata applicata la pratica di **Continuous Integration** (CI), ossia l'allineamento frequente (ovvero "molte volte al giorno") dagli ambienti di lavoro degli sviluppatori che

lavorano sullo stesso progetto verso l'ambiente condiviso. In questo caso, ovviamente, lo sviluppatore è solo uno. Comunque, è stato utilizzato GitHub come servizio di hosting del progetto e Gitlab per la pratica di Continuous Integration. Ogni volta che viene eseguito un push sulla repository remota di GitHub, la repository "mirror" di GitLab compila il progetto ed esegue i test di unità, notificando il risultato finale di entrambe le operazioni.

Status	Pipeline	Triggerer	Stages	
 passed 00:00:53 3 hours ago	Modifiche pom #928712160  main  3302914b  latest		 	
 failed 00:00:56 3 hours ago	Tradotta ASMETA vascaidro in java #928670994  main  c9917a79 		 	 
 passed 00:00:52 21 hours ago	Implementata classe StazioneMeteo e ASMETA vas... #927852185  main  62c455d8 		 	
 passed 00:00:51 1 day ago	Implementazione StazioneMeteo #927589411  main  88a81870 		 	
 passed 00:00:53 1 day ago	Merge branch 'main' of https://github.com/Pellegrin... #926440289  main  ddc75031 		 	

Verification

Sulla classe StazioneMeteo è stata fatta anche **verifica del codice** attraverso **Design By Contract (DBC)**, ossia una metodologia di progettazione software che prevede di scrivere per ogni metodo e costruttore dei contratti che indicano gli obblighi del "cliente" (chi chiama il metodo) e del "fornitore" (che scrive il metodo). Il tool utilizzato è **OpenJML** con solver Z3-4.3.2, che a partire dai contratti e dall'implementazione dei metodi fornisce una verifica formale del codice. Tutti i metodi ed il costruttore della classe StazioneMeteo sono stati validati eccetto che per il metodo che calcola l'umidità media (i cui contratti utilizzano una feature di OpenJML non ancora completamente implementate, ossia il costrutto `\sum`).

```

  ✓ Static Checks for: SmartHome
    ✓ smart_home_additional_elements
      ✓ smart_home_additional_elements.StazioneMeteo
        [VALID] allerta(double,int,double,double) [1,153 z3_4_3]
        [VALID] calcoloDewPoint(double,int) [1,155 z3_4_3]
        [VALID] getMaxTemp() [1,152 z3_4_3]
        > [INVALID] getUmiditaMedia() [1,137 z3_4_3]
        [VALID] StazioneMeteo(double[],int[]) [1,249 z3_4_3]

```

Inoltre, nell'ambito della verifica del codice, è stato utilizzato il tool **SpotBugs** per l'**analisi statica** volta ad individuare errori e bad practice nel codice. Gli errori trovati sull'intero progetto sono tre, di cui due dello stesso tipo:

```

  ✓ > SmartHome (3) [ProgettoTVSW main]
    ✓ Of Concern (3)
      ✓ Normal confidence (3)
        ✓ May expose internal representation by incorporating reference to mutable object (2)
          ✗ new smart_home_additional_elements.StazioneMeteo(double[], int[]) may expose internal representation by storing an externally mutable object into StazioneMeteo.templInterna [Of Concern(18), Normal confidence]
          ✗ new smart_home_additional_elements.StazioneMeteo(double[], int[]) may expose internal representation by storing an externally mutable object into StazioneMeteo.umiditaInterna [Of Concern(18), Normal confidence]
        ✓ Integral division result cast to double or float (1)
          ✗ Integral division result cast to double or float in smart_home_additional_elements.StazioneMeteo.calcoloDewPoint(double, int) [Of Concern(17), Normal confidence]

```

Tutti e tre sono indicati di concern basso (18 e 17 su un massimo di 20, dove 20 è il livello di “preoccupazione” minimo) e sono stati conseguentemente rimossi.

Infine, nel corso dell’implementazione dell’intero progetto sono state effettuate periodicamente le attività di **code inspection** (ispezione “manuale” del codice) e **code refactoring** (ad esempio rendendo più chiari i nomi dei vari artefatti software, ma anche aggiustando i problemi individuati da SpotBugs, legati più alle good practice che a veri e propri bug).

Modeling

Come già indicato, sono stati modellati due aspetti del sistema, ovvero la smart home e la vasca idromassaggio. Per farlo si è utilizzato il **framework ASMETA**, che permette principalmente di modellare una macchina ASM (Abstract State Machine), simularne il comportamento (anche attraverso un animatore) e scrivere dei casi di test (scenari AVALLA). Di seguito vengono riportati degli **esempi di animazione** delle due macchine:

smarthome.asm

Type	Functions	State 0	State 1	State 2	State 3	State 4	State 5	State 6
<input type="checkbox"/> M	elemento	CASA	CASA	LUCI	TAPPARELLE	CASA	PORTE	
<input type="checkbox"/> M	azioneCasa	APRI_TUTTO	APRI_TUTTO	APRI_TUTTO	APRI_TUTTO	CHIUDI_TUTTO	CHIUDI_TUTTO	
<input type="checkbox"/> C	statoTapparella(3)	100	0	100	100	37	100	100
<input type="checkbox"/> C	statoPorta(GARAGE)	CHIUSA	APERTA	CHIUSA	CHIUSA	CHIUSA	CHIUSA	CHIUSA
<input type="checkbox"/> C	statoTapparella(2)	100	0	100	100	100	100	100
<input type="checkbox"/> C	statoTapparella(1)	100	0	100	100	100	100	100
<input type="checkbox"/> C	statoLuce(4)	SPENTA	SPENTA	ACCESA	ACCESA	ACCESA	ACCESA	ACCESA
<input type="checkbox"/> C	statoLuce(3)	SPENTA	SPENTA	ACCESA	SOFFUSA	SOFFUSA	SOFFUSA	SOFFUSA
<input type="checkbox"/> C	statoLuce(5)	SPENTA	SPENTA	ACCESA	ACCESA	ACCESA	ACCESA	ACCESA
<input type="checkbox"/> C	statoPorta(PRINCIPALE)	CHIUSA	APERTA	CHIUSA	CHIUSA	CHIUSA	CHIUSA	APERTA
<input type="checkbox"/> C	statoPorta(RETRO)	CHIUSA	APERTA	CHIUSA	CHIUSA	CHIUSA	CHIUSA	CHIUSA
<input type="checkbox"/> C	statoLuce(2)	SPENTA	SPENTA	ACCESA	ACCESA	ACCESA	ACCESA	ACCESA
<input type="checkbox"/> C	statoAntifurto	true	true	true	true	true	true	true
<input type="checkbox"/> C	statoLuce(1)	SPENTA	SPENTA	ACCESA	ACCESA	ACCESA	ACCESA	ACCESA
	azioneLuci			LUCE_SOFFUSA	LUCE_SOFFUSA	LUCE_SOFFUSA	LUCE_SOFFUSA	
	luce			3	3	3	3	
	azioneTapparella				IMPOSTA_TAPPARELLA	IMPOSTA_TAPPARELLA	IMPOSTA_TAPPARELLA	
	livello_tapparella				37	37	37	
	tapparella				3	3	3	
	azionePorte						APRI_PORTA	
	porta						PRINCIPALE	

vascaidro.asm

Type	Functions	State 0	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8	State 9
<input type="checkbox"/> M	riempi_completamente	true	false	false	false	true	false	true	false	false	
<input type="checkbox"/> C	statoLivello		100	0	25	50	100	75	100	100	100
<input type="checkbox"/> M	svuota_completamente		true	false	false	false	false	false	false	false	
<input type="checkbox"/> M	riempi_25_percento			true	true	true	false	false	true	false	
<input type="checkbox"/> M	svuota_25_percento						true	true	true	false	

Per la prima macchina è stato scritto uno **scenario** (per la validazione della macchina), tradotto poi in un metodo JUnit, e sono state provate le seguenti proprietà con il **model checker**:

- In qualsiasi momento, è possibile avere una luce accesa in un qualche stato futuro (proprietà di Liveness);
- Non è mai possibile avere la porta principale chiusa e l'antifurto disattivato (proprietà di Safety)
- Se l'antifurto non è attivo, non è possibile che si accenda più di una luce alla volta (proprietà di Safety)
- Ogni luce accesa rimane accesa finché non si opera direttamente sulla luce.

Non è stato possibile eseguire il model advisor per la verifica delle proprietà statiche della macchina ed il tool ATGT per la generazione automatica dei casi di test (scenari AVALLA) a causa della complessità del modello.

Essendo invece il secondo modello molto più semplice, è stato possibile utilizzare i tool **MA (model advisor)** e **ATGT**. La AMS è stata implementata in modo semplice e con pochi costrutti con l'obiettivo di poter usare tali tool. Tutte le sette proprietà statiche del model advisor sono passate eccetto la seconda (completezza delle conditional rule). Mentre ATGT ha generato un totale di 8 casi di test sotto forma di scenari avalla, che sono poi stati tradotti in metodi JUnit che hanno garantito una copertura del 100% di statement, branch e term.

Model-Based Testing

È stato modellato il dominio di input (**input domain modeling**) del metodo di calcolo del dew point della classe StazioneMeteo:

```
public double calcoloDewPoint(double T, int RH) {
    double Tmin = -90, Tmax = 60; //Minima (e massima) temperatura
    rilevata sulla Terra
    if (T < Tmin || T > Tmax || RH <= 0 || RH > 100)
        return Double.NaN;
    return T - (double)(100 - RH) / 5;
}
```

Attraverso l'approccio basato sulle funzionalità (**functional based approach**) sono stati partizionati i due domini come segue:

double T	
$T < T_{min}$	
$T_{min} \leq T < 0$	
$T = 0$	
$0 < T \leq T_{max}$	
$T > T_{max}$	$T_{min} = -90^{\circ}\text{C} \qquad T_{max} = 60^{\circ}\text{C}$

int RH	
$RH < 0$	
$RH = 0$	
$0 < RH < 100$	
$RH = 100$	
$RH > 100$	

In questo caso la tecnica di **combinatorial testing pair wise** coincide con la tecnica **ACoC** (All Combinations) e produce 25 casi di test (dati da tutte le possibili combinazioni delle due tabelle) che portano a una copertura del 100% di statement, branch e term. Come già detto, per gestire 25 input diversi è stato utilizzato il testing parametrico.

Infine, anche la **generazione automatica dei casi di test** a partire dalla ASM attraverso il tool **ATGT** è una tecnica di model-based testing e, come già detto, ha permesso di ottenere una copertura del 100% di statement, branch e term una volta che gli scenari sono stati tradotti in codice JUnit.