Software Systems Engineering: Final task 2018

Filippo Frabetti, Nicola Semprini, Paolo Magnani

Alma Mater Studiorum – University of Bologna
viale Risorgimento 2, 40136 Bologna, Italy
filippo.frabetti@studio.unibo.it
nicola.semprini4@studio.unibo.it
paolo.magnani5@studio.unibo.it

Abstract. This document is the explicit representation of the production process adopted for the Software Systems Engineering final task 2018.

Table of Contents

1 Introduction

L'ordine di presentazione delle sezioni in questo documento è solo parzialmente cronologico, in quanto spesso, durante il processo di sviluppo, è stato necessario rivedere le scelte fatte in precedenza alla luce di nuove considerazioni.

2 Vision

In fase di analisi e di progettazione scegliamo di procedere top-down, con zooming progressivo verso i dettagli.

Il processo di sviluppo adottato è di tipo *Agile*, ovvero prediligendo ad ogni step la creazione di prototipi funzionanti da sottoporre al committente al fine di ottenere dei feedback.

In accordo con questa visione, anche i requisiti vengono affrontati in maniera incrementale.

3 Requirements

In a home of a given city (e.g. Bologna), a `ddr` robot is used to clean the floor of a room (`R-FloorClean`).

The floor in the room is a flat floor of solid material and is equipped with two *sonars*, named `sonar1` and `sonar2` as shown in the picture [1] (`sonar1` is that at the top). The initial position (`start-point`) of the robot is detected by `sonar1`, while the final position (`end-point`) is detected by `sonar2`.

The robot works under the following conditions:

1. `R-Start`: an `authorized user` has sent a `START` command by using a human `GUI` interface (`console`) running on a conventional PC or on a smart device (`Android`).
2. `R-TempOk`: the value temperature of the city is not higher than a prefixed value (e.g. `25` degrees Celsius).
3. `R-TimeOk`: the current clock time is within a given interval (e.g. between `7` a.m and `10` a.m).

While the robot is working:

- it must blink a `Led` put on it, if the robot is a `real` robot (`R-BlinkLed`).
- it must blink a `Led Hue Lamp` available in the house, if the robot is a `virtual` robot (`R-BlinkHue`).
- it must avoid fixed obstacles (e.g. furniture) present in the room (`R-AvoidFix`) and/or mobile obstacles like balls, cats, etc. (`R-AvoidMobile`).

[1] *iss2018\it.unibo.issMaterial\issdocs\Material\nodeLab2018.pdf*: **8 Final task 2018**

Moreover, the robot must stop its activity when one of the following conditions apply:

1. R-Stop: an authorized user has sent a STOP command by using the console.
2. R-TempKo: the value temperature of the city becomes higher than the prefixed value.
3. R-TimeKo: the current clock time is beyond the given interval.
4. R-Obstacle: the robot has found an obstacle that it is unable to avoid.
5. R-End: the robot has finished its work.

4 Requirements analysis

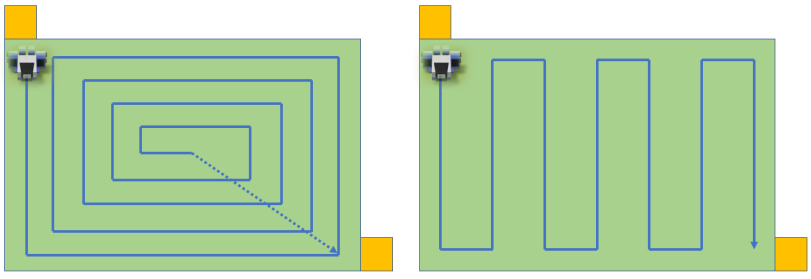
Per prima cosa analizziamo i termini che compaiono nei requisiti allo scopo di comprenderne il significato e le problematiche. Il passo successivo è la definizione dell'architettura che deriva dall'analisi dei requisiti e la sua conseguente formalizzazione.

4.1 Cleaning the room

Pulire la stanza significa trovare un percorso, da un punto di partenza (start-point) ad un punto di arrivo (end-point) predefiniti, che copra l'intera superficie disponibile. Ciò può essere fatto in svariati modi, tuttavia si considera desiderabile impiegare il minor tempo possibile e quindi limitare il passaggio ripetuto per le stesse posizioni già pulite in precedenza.

Consideriamo inizialmente una stanza rettangolare e priva di ostacoli. In questo caso esistono più percorsi che consentono di non passare mai due volte per lo stesso punto: ad esempio procedendo a spirale lungo le pareti e proseguendo verso il centro della stanza, oppure a "zig-zag" parallelamente ad una parete, avvicinandosi via via a quella opposta.

La differenza sostanziale tra le due risiede nella posizione finale: nella prima questa si trova al centro della stanza, mentre nella seconda lungo la parete opposta a quella di partenza. Da notare come nella seconda alternativa, data la specifica della stanza e la collocazione dei due sonar fissi, il punto di arrivo coincida con sonar2.



L'assunzione relativa alla forma rettangolare della stanza non è assolutamente restrittiva, mentre la presenza di eventuali ostacoli riguarda i requisiti trattati di seguito.

4.1.1 R-End

Il robot deve fermarsi quando ha terminato la pulizia dell'intera stanza. Ciò significa, in altre parole, aver percorso almeno una volta tutti i punti della superficie. Ci chiediamo come sia possibile verificare questo requisito.

Se conoscessimo le dimensioni della stanza, basterebbe tenere traccia del percorso seguito per controllare la superficie coperta dal robot. Un modo per fare ciò potrebbe essere rappresentare la stanza come una scacchiera composta da un reticolo di celle quadrate che il robot può percorrere con una mossa elementare (un "passo"). La dimensione delle celle deve essere sufficientemente piccola per essere vicina a un numero divisibile per le due dimensioni della stanza, così da limitare la presenza di "mezzi-passi", ma anche abbastanza grande da essere misurabile dal robot.

L'unico modo che il robot ha a disposizione per misurare le distanze è tramite il tempo, ovvero, assumendo di muoversi a velocità costante (trascurando l'accelerazione dovuta alla partenza e allo stop), lo spazio percorso è dato dalla legge:

$$s = vt$$

Il robot quindi, muovendosi all'interno della griglia, può tenere traccia di tutti i passi fatti così da poter verificare in seguito di essere passato almeno una volta su ogni casella. Questo comporta però di conoscere prima la dimensione della stanza.

In alternativa, adottando un movimento a "zig-zag", quando il robot viene rilevato molto vicino a `sonar2` possiamo ragionevolmente assumere che questi abbia pulito l'intera stanza. Ciò è sicuramente vero nel caso di una semplice stanza rettangolare e priva di ostacoli, la cui lunghezza è multiplo pari del "passo laterale" compiuto dal robot al termine di ogni rettilineo.

Nel caso di multiplo dispari, l'ultimo tratto verrebbe percorso due volte: una per effettuare la pulizia e l'altra per raggiungere la posizione di `sonar2`.

4.1.2 R-AvoidFix, R-AvoidMobile and R-Obstacle

Possiamo dividere gli ostacoli in due categorie: fissi e mobili. I primi vanno necessariamente aggirati in qualche modo, mentre per i secondi il robot può aspettare che l'ostacolo si sposti per poi proseguire il proprio percorso di pulizia.

Inoltre, ci sono alcuni tipi di ostacoli che il robot potrebbe essere incapace di superare (`R-Obstacle`), come ad esempio una campana posta su di esso o una barriera imprevista da parete a parete che separa in due la stanza.

Come analisti riteniamo però che anche gli ostacoli il cui tentato superamento comporti un numero di fallimenti superiore ad una certa soglia vadano

considerati invalicabili. Dal punto di vista del robot risulterebbe infatti complesso stabilire quando ha a che fare con un ostacolo "effettivamente" insuperabile (come nell'esempio della campana) o con uno molto difficile da superare, ovvero che porti al fallimento di numerosi tentativi di superamento.

4.2 R-Start and R-Stop

Per avviare la pulizia della stanza, è necessario l'invio dell'apposito comando da parte di un utente autorizzato che si interfaccia al sistema mediante una `console`. Tale utente può anche richiedere la terminazione prematura dell'operazione.

Un utente è autorizzato se è dotato di un username e una password con cui accedere alla console, pertanto è necessario offrire un servizio di login che, supportato da un meccanismo per la persistenza dei dati, permetta di ottenere delle credenziali per accedere al sistema.

Inoltre, i requisiti riferiscono esplicitamente che l'interfaccia grafica sia utilizzabile da remoto da qualsiasi dispositivo, ad esempio da PC o smartphone. Se la console fosse raggiungibile via web, qualsiasi utente umano potrebbe pilotare il robot tramite un browser, purché precedentemente registrato e autenticato nel sistema.

4.3 Additional requirements

- `R-TimeOk`/`R-TimeKo`: il robot può svolgere il suo lavoro solo durante un certo intervallo temporale, determinato da un orologio presente nel sistema (ad esempio quello della macchina locale).
- `R-TempOk`/`R-TempKo`: per poter utilizzare il robot la temperatura non può essere superiore ad una certa soglia, quindi il sistema deve essere in grado di interagire con un sensore remoto per ricevere i dati termici aggiornati.
- `R-BlinkLed`: durante il movimento, il robot fisico deve far lampeggiare un led posto su di esso.
- `R-BlinkHue`: il robot virtuale deve far lampeggiare una Hue Lamp Philips: una *smart thing* con cui è possibile interagire da remoto.

I componenti richiesti da questi requisiti sono definiti formalmente nella sezione successiva.

4.4 System architecture

Dai requisiti emerge il fatto che il sistema sia distribuito e composto da due nodi di esecuzione: uno per il robot ed uno che ospita la logica di controllo e fornisce la console, usabile da remoto, per pilotare il robot.

Si delineano nello specifico 5 entità:

1. il robot, che può essere sia fisico sia virtuale (`RealRobot` e `VirtualRobot`), e l'ambiente in cui si muove (fisico/virtuale)
2. gli attuatori: il `Led` per il robot reale e l'`HueLamp` per quello virtuale
3. i sensori: un indicatore di temperatura remoto (`RemoteThermometer`) e un orologio (`ClockAgent`)
4. un utilizzatore umano (`HumanOperator`) a cui è collegata la console grafica
5. un componente che andrà ad incapsulare l'`Application logic`

L'ambiente in cui si trova il robot è inoltre costituito da:

- 2 sensori: `sonar1` e `sonar2`
- ostacoli fissi (pareti) e mobili

L'architettura del sistema può essere quindi descritta in maniera informale dalla Fig. 1.

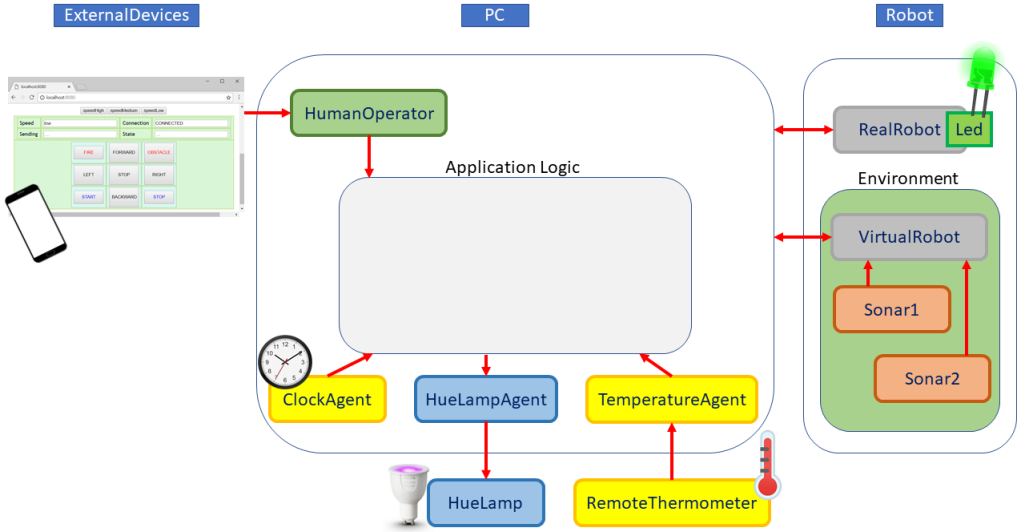


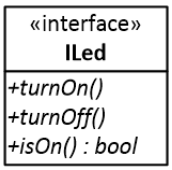
Fig. 1. Architettura informale ottenuta dall'analisi dei requisiti

Passiamo ora ad analizzare formalmente i singoli componenti presenti all'interno dei due contesti, a partire da quello che rappresenta l'applicazione principale.

4.5 Actuators

Il `Led`, essendo un componente estremamente semplice e trovandosi sul robot fisico, può essere visto dal punto di vista dell'applicazione come un'entità passiva

senza un proprio flusso di controllo. Pertanto, il modello del `Led` – un oggetto Java – può essere formalmente definito dalla seguente interfaccia:



L'interazione con il `Led` può quindi avvenire tramite *procedure call* ad opera del robot fisico su cui si trova.

Il lampeggiamento del led dovrà essere gestito da un apposito componente attivo il cui compito è accenderlo e spegnerlo ripetutamente, in base alla logica applicativa, utilizzando i metodi esposti dall'interfaccia.

L'`Hue Lamp` è invece un'entità attiva che ci viene fornita dal committente insieme ad un bridge per poter interagire con essa attraverso un'interfaccia *RESTful* [2].

Possiamo quindi definire il modello di un attore in grado di interagire con l'`Hue Lamp` remota: per farlo usiamo il linguaggio custom della nostra Software House *QActor*, poiché ci permette di modellare sistemi distribuiti eterogenei.

```
18 // Event lightCmd : lightCmd(CMD)
19 QActor huelampagentra context ctxReqAnalysis {
20         Plan init normal [
21                 println("hueLampAgent start")
22         ]
23         switchTo waitForEvents
24
25         Plan waitForEvents [
26                 println("hueLampAgent waiting for events")
27         ]
28         transition stopAfter 3600000
29                 whenEvent lightCmd -> commandLamp
30         finally repeatPlan
31
32         Plan commandLamp resumeLastPlan [
33                 onEvent lightCmd : lightCmd(blink) -> javaRun it.unibo.
                        finalTask2018.ra.hueAdapter.setLight("blink");
34                 onEvent lightCmd : lightCmd(on)  -> javaRun it.unibo.
                        finalTask2018.ra.hueAdapter.setLight("on");
35                 onEvent lightCmd : lightCmd(off) -> javaRun it.unibo.
                        finalTask2018.ra.hueAdapter.setLight("off")
36         ]
37 }
```

[2] `https://www.developers.meethue.com/philips-hue-api`

Listing 1.1. ../it.unibo.finalTask2018/src/requirementsAnalysis.qa

Si noti come la scelta di utilizzare l'evento `lightCmd` consenta di controllare più dispositivi senza che questi siano noti a priori dalla sorgente che emette tali eventi in base alla logica applicativa.

Inoltre, poiché i comandi utilizzati nel payload dell'evento non si limitano al solo *lampeggiamento*, il modello è generale e facilmente riutilizzabile. Ad esempio un cambiamento dei requisiti potrebbe stabilire che la lampada debba restare acceso durante il movimento del robot.

Dal momento che l'implementazione del comportamento technology-dependent è delegata a metodi Java (`javaRun`), un modello analogo può essere utilizzato per il lampeggiamento del led.

4.6 Sensors

4.6.1 TemperatureAgent

Poiché abbiamo a che fare con un servizio remoto che fornisce i dati sulla temperatura, modelliamo un attore che funga da *adapter* e permetta la comunicazione tra questi e il sistema.

In generale, l'applicazione può interagire con l'adapter mediante tre possibili approcci del tipo produttore/consumatore:

- **polling:** chi necessita dell'informazione si fa carico di fare richiesta periodicamente al produttore;
- **pattern observer:** l'observer si registra presso l'observable e viene notificato al cambiamento di stato di quest'ultimo;
- **publish/subscribe:** è il produttore stesso a pubblicare all'esterno le informazioni quando queste sono disponibili, ad uso degli eventuali consumatori.

Scegliamo di adottare la terza strategia, poiché meno costosa in termini di dati scambiati e poiché garantisce un minor accoppiamento tra le entità coinvolte.

Nell'ambiente distribuito in cui operano i QActor, la pubblicazione (*publish*) di informazioni avviene tramite l'emissione di **eventi** che per essere rilevati necessitano che il consumatore si sia preventivamente messo in ascolto (operazione logicamente equivalente alla *subscribe*).

```
41 // Event temperature : temperature(T)
42 QActor temperatureagentra context ctxReqAnalysis {
43        Plan init normal [
44               println("temperatureAgent start")
45        ]
46        switchTo checkTemperature
47
```

```
48          Plan checkTemperature [
49                  delay 5000;
50                  println("temperatureAgent checking temperature");
51                  javaRun it.unibo.finalTask2018.ra.temperatureAdapter.
                        getTemperature();
52                  [ ?? currentTemp(T) ] emit temperature : temperature(T)
53          ]
54          finally repeatPlan
55  }
```

Listing 1.2. ../it.unibo.finalTask2018/src/requirementsAnalysis.qa

Come per `lightCmd`, anche in questo caso l'evento `temperature` emesso dall'adapter del sensore può essere percepito da chiunque sia in ascolto, senza dover specificare a priori i possibili componenti interessati.

Si noti infine come l'interazione tra `TemperatureAgent` e il servizio che fornisce la temperatura sia a polling.

4.6.2 ClockAgent

Per quanto riguarda il `Clock`, scegliamo di modellarlo come un attore a se stante, separato dalla logica applicativa specifica del problema, così che sia facilmente modificabile e riutilizzabile.

Il modello di tale attore è pressoché analogo a quello visto per `TemperatureAgent`.

```
59  // Event clock : clock(H,M)
60  QActor clockagentra context ctxReqAnalysis {
61          Plan init normal [
62                  println("clockAgent start")
63          ]
64          switchTo checkTime
65
66          Plan checkTime [
67                  delay 6000; // for testing
68                  // delay 60000;
69                  println("clockAgent checking time");
70                  javaRun it.unibo.finalTask2018.ra.clockAdapter.getTime();
71                  [ ?? currentTime(H,M) ] emit clock : clock(H,M)
72          ]
73          finally repeatPlan
74  }
```

Listing 1.3. ../it.unibo.finalTask2018/src/requirementsAnalysis.qa

4.7 HumanOperator

Lo `HumanOperator` può essere modellato come un emettitore di comandi per il robot. Per disaccoppiare le comunicazioni tra sorgente e destinatario dei comandi, l'operator si rivolge ad una terza entità che rappresenta la logica applicativa, così da non essere vincolato a conoscere l'identità di ogni possibile robot pilotabile.

Si noti come tale disaccoppiamento funziona anche al contrario: gli eventuali robot possono essere pilotati il maniera uniforme da molteplici sorgenti di comandi, tutte facenti riferimento alla stessa application logic.

```
78 // Dispatch cmd : cmd(CMD)
79 QActor humanoperatorra context ctxReqAnalysis {
80         Plan init normal [
81                 println("humanOperator start")
82         ]
83         switchTo sendCmd
84 
85         Plan sendCmd [
86                 // for testing
87                 delay 800;
88                 forward applra -m cmd : cmd(w(X));
89                 delay 800;
90                 forward applra -m cmd : cmd(h(X))
91         ]
92 }
```

Listing 1.4. ../it.unibo.finalTask2018/src/requirementsAnalysis.qa

In questa prima fase di modellazione, lo `HumanOperator` va a generare programmaticamente i comandi per il robot. In un secondo momento questi saranno invece originati da un effettivo utente umano che utilizza una console grafica (Subsection 5.5).

4.8 Application logic

Per poter attuare la logica applicativa stabilita dai requisiti, l'applicazione deve ricevere i dati della temperatura, del tempo e dei vari sonar, emessi sotto forma di eventi; oltre a questi, deve mettersi in attesa dei comandi di movimento inviati dallo `HumanOperator`, così da poterli inoltrare a tutti i robot disponibili (reali e virtuali): a tal fine è stato introdotto l'evento `robotCmd : moveRobot(CMD)`.

Sempre in accordo con i requisiti, l'applicazione deve gestire ed emettere, quando opportuno, gli eventi contenenti i comandi che regolano il `Led`, presente sul robot fisico, e la `Hue Lamp` remota, gestita da quello virtuale.

```
96 QActor applra context ctxReqAnalysis {
97         Plan init normal [
```

```
98              println("appl start")
99      ]
100     switchTo waitForCmd
101
102     Plan waitForCmd [ ]
103     transition stopAfter 3600000
104             whenEvent temperature : temperature(T) do printCurrentEvent
                    ,
105             whenEvent clock : clock(H,M) do printCurrentEvent,
106             whenEvent sonarSensor : sonar(N,D) do printCurrentEvent,
107             whenEvent frontSonar : sonar(DISTANCE) do printCurrentEvent
                    ,
108             whenMsg cmd -> msgReceived
109     finally repeatPlan
110
111     Plan msgReceived resumeLastPlan [
112             printCurrentMessage;
113             onMsg cmd: cmd(CMD) -> emit robotCmd : moveRobot(CMD);
114
115             // for testing
116             onMsg cmd: cmd(w(X)) -> emit lightCmd : lightCmd(on);
117             onMsg cmd: cmd(h(X)) -> emit lightCmd : lightCmd(off)
118     ]
119 }
```

Listing 1.5. ../it.unibo.finalTask2018/src/requirementsAnalysis.qa

4.9 Robot

Ci viene fornito un modello del robot espresso nel linguaggio *QActor*[3]: questi è in grado di ricevere messaggi `moveRobot : moveRobot(CMD)` e di interpretarli come comandi di movimento (forward, backward, left, right, stop).

Il robot è anche dotato di un sonar frontale che gli permette di rilevare la presenza di ostacoli (`R-AvoidFix` e `R-AvoidMobile`), rivestendo il duplice ruolo di attuatore e di sensore, in quanto emette informazioni relative alla distanza dagli ostacoli che incontra sotto forma di eventi `frontSonar : sonar(DISTANCE)`.

Il robot è quindi un'entità attiva il cui scopo è interpretare i messaggi ricevuti in un certo formato ed emettere eventi relativi al sonar frontale.

Poiché l'attore che incarna la logica applicativa emette comandi mediante eventi `robotCmd`, è necessario introdurre un ulteriore componente che effettui il mapping di questi nei rispettivi messaggi `moveRobot` gestiti dal robot. In questo caso, dal momento che vogliamo evitare che i comandi per pilotare il robot vadano persi, facciamo ricorso per tale scopo ad un componente *event-driven*

[3] iss2018Lab/it.unibo.mbot.divide/src/realRobotExecutor.qa e virtualRobotExecutor.qa

(EventHandler).

```
System finalTask2018

Dispatch moveRobot : moveRobot(CMD)
Event robotCmd : moveRobot(CMD)
Event lightCmd : lightCmd(CMD)
Event frontSonar : sonar(DISTANCE)
Event sonarSensor : sonar(NAME,DISTANCE)

Context ctxDdr ip [ host="localhost" port=8882 ]
EventHandler evhddr for robotCmd {
       forwardEvent ddr -m moveRobot // event -> msg (stesso payload)
};
EventHandler evhddrlogger for sonarSensor, frontSonar, robotCmd, lightCmd
     /*-print*/ {
       memoCurrentEvent -lastonly for ddrlogger;
       demo dologevent for ddrlogger
};

// see: raintegrator.qa
//Context ctxReqAnalysis ip [ host="localhost" port=8888 ] -standalone

QActor ddr context ctxDdr {
       Plan init normal [
              println("ddr start");
              javaRun it.unibo.finalTask2018.ra.robotAdapter.
                  setUpEnvironment()
       ]
       switchTo waitForCmd

       Plan waitForCmd [
              println("ddr waiting for commands")
       ]
       transition stopAfter 3600000
              whenMsg moveRobot -> execMove
       finally repeatPlan

       Plan execMove resumeLastPlan [
              onMsg moveRobot : moveRobot( h(X) ) -> javaRun it.unibo.
                  finalTask2018.ra.robotAdapter.robotStop();
              onMsg moveRobot : moveRobot( w(X) ) -> javaRun it.unibo.
                  finalTask2018.ra.robotAdapter.robotForward();
              onMsg moveRobot : moveRobot( s(X) ) -> javaRun it.unibo.
                  finalTask2018.ra.robotAdapter.robotBackward();
              onMsg moveRobot : moveRobot( a(X) ) -> javaRun it.unibo.
                  finalTask2018.ra.robotAdapter.robotLeft();
              onMsg moveRobot : moveRobot( d(X) ) -> javaRun it.unibo.
                  finalTask2018.ra.robotAdapter.robotRight()
```

```
41          ]
42  }
```

Listing 1.6. ../it.unibo.finalTask2018/src/ddr.qa

L'implementazione delle varie mosse viene delegata ad un'opportuna classe Java, così che questa possa essere modificata senza alterare il modello del robot. Questa soluzione permette inoltre di riutilizzare lo stesso modello per entrambe le tipologie di robot, reale e virtuale.

Nella stanza sono presenti due sonar in grado di rilevare la presenza di un robot che passi in corrispondenza di uno dei due, emettendo in tal caso un evento `sonarSensor : sonar(NAME,DISTANCE)`, dove `NAME` è `sonar1` o `sonar2`.

L'emissione di tali eventi dipende dalla specifica implementazione dell'*environment*, pertanto anch'essa viene delegata ad una classe Java:

```
9           public static void setUpEnvironment(QActor qa) {
10                  new Thread(() -> {
11                          try {
12                                  Thread.sleep(1000);
13                                  qa.emit("sonarSensor", "sonar(sonar1,8)");
14                                  System.out.println("Emitting sonarSensor :
                                        sonar(sonar1,8)");
15
16                                  Thread.sleep(1000);
17                                  qa.emit("frontSonar", "sonar(10)");
18                                  System.out.println("Emitting frontSonar :
                                        sonar(10)");
19                          } catch (InterruptedException e) { }
20                  }).start();
21          }
```

Listing 1.7. ../it.unibo.finalTask2018/src/it/unibo/finalTask2018/ra/robotAdapter.java

4.10 System formal definition

L'immagine della Fig. 1 può essere quindi formalmente definita dal sistema *QActor* risultante dall'integrazione tra `ddr.qa` e `requirementsAnalysis.qa`, rappresentata da un ulteriore componente il cui compito è mettere in contatto i contesti degli altri due.

```
1   System raintegrator
2
3   Event clock : clock(H,M)    // H = ore ([0-23]), M = minuti ([0-59])
4   Event temperature : temperature(T)  // T = temperatura in celsius (float)
```

```
5  Event frontSonar : sonar(DISTANCE)  // DISTANCE = distanza dell'ostacolo
       (int)
6  Event sonarSensor : sonar(NAME,DISTANCE) // NAME = sonar1 | sonar2
7  Event lightCmd : lightCmd(CMD)   // CMD = on | off | blink
8  Event robotCmd : moveRobot(CMD)   // CMD = C(X), C = w|a|s|d|h, X =
       velocita'
9  Dispatch cmd : cmd(CMD)
10
11 Context ctxRaIntegrator ip [ host="localhost" port=8880 ]
12
13 Context ctxDdr ip [ host="localhost" port=8882 ] -standalone
14 Context ctxReqAnalysis ip [ host="localhost" port=8888 ] -standalone
```

Listing 1.8. ../it.unibo.finalTask2018/src/raintegrator.qa

In alternativa, è possibile decommentare in `ddr.qa` la riga contenente la conoscenza del contesto dell'applicazione principale, aggiungendo quindi il vincolo che al momento dell'esecuzione del robot questa sia già attiva.

Questa seconda opzione è meno conveniente dal punto di vista del testing automatizzato dei singoli componenti, discusso nella Subsection 8.1.

Poiché abbiamo utilizzato un *modello eseguibile*, questo ci fornisce anche un primo **prototipo** funzionante della nostra applicazione.

5 Problem analysis

Il problema definito dai requisiti si colloca nell'ambito dell'IoT, dove è diffuso l'utilizzo della cosiddetta *architettura esagonale* come design pattern per gestire il rapporto tra input, elaborazione ed output nei sistemi distribuiti. In tale pattern, un ruolo centrale viene rivestito dalla *business logic* e dal modello delle risorse (*resource model*), mentre gli altri componenti del sistema vengono visti tramite degli adapter verso le varie tecnologie ed implementazioni.

Il modello delle risorse è quindi un disaccoppiatore del mondo fisico da quello della logica applicativa, che deve essere indipendente dalla tecnologia. Nello specifico:

1. un cambiamento dello stato del sensore fisico causa un cambiamento dello stato del modello del sensore stesso
2. al cambiamento del modello, l'informazione viene notificata al controller (è un observer del modello)
3. il controller modifica quindi il modello dell'attuatore in base alla logica applicativa
4. l'aggiornamento del modello dell'attuatore comporta infine un aggiornamento dell'attuatore fisico

Come analisti riteniamo vantaggioso introdurre un modello delle risorse così da non vincolare la logica applicativa a dettagli tecnologico-implementativi,

sebbene ciò comporti un piccolo sforzo aggiuntivo iniziale. Tale overhead è però un investimento a fronte dello sviluppo futuro di applicazioni simili.

5.1 Resource Model

Per la definizione del modello delle risorse usiamo un linguaggio dichiarativo che consenta di esprimere sia lo stato di ciascun componente, sia le operazioni primitive su di esso. A tal fine, introduciamo il seguente modello scritto in Prolog:

```
% fatti
limitTemperatureValue(35).
timeInterval(h(7,0),h(10,0)).

% modello di sensori e attuatori
model( type(actuator, led), name(l1), value(off) ).
model( type(sensor, temperature), name(t1), value(20) ).
model( type(sensor, clock), name(c1), value(h(8,0)) ).

% modello del robot
model( type(robots, robot), name(r1), value(h(0)) ).

% modello del software agent
model( type(software, agent), name(swag), value(idle) ).

% -------- OPERAZIONI SUL MODELLO --------
getModelItem( TYPE, CATEG, NAME, VALUE ) :-
       model( type(TYPE, CATEG), name(NAME), value(VALUE) ).

getModelItem( NAME, VALUE ) :-
       model( _, name(NAME), value(VALUE) ).

% se devo sovrascrivere lo stesso valore, non faccio niente
changeModelItem( CATEG, NAME, VALUE ) :-
       getModelItem( _, CATEG, NAME, VALUE ), !.

changeModelItem( CATEG, NAME, VALUE ) :-
       replaceRule(
              model( type(TYPE, CATEG), name(NAME), value(_) ),
              model( type(TYPE, CATEG), name(NAME), value(VALUE) )
       ), !,
       % output( changedModelAction(CATEG, NAME, VALUE) ),
       ( % to be defined by the appl designer (in Rules)
              changedModelAction(CATEG, NAME, VALUE); % or
              true  % to avoid failure if no changedModelAction is
                  defined
       ).
```

Listing 1.9. ../it.unibo.finalTask2018/resourceModel.pl

5.2 Logical architecture

In seguito all'introduzione del modello delle risorse possiamo definire l'architettura logica del sistema in modo informale tramite la Fig. 2.

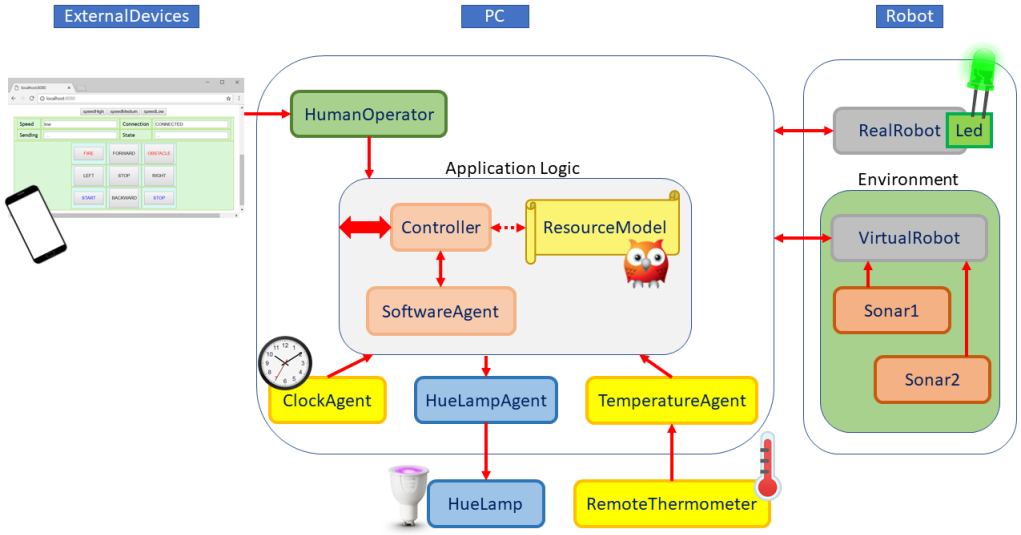


Fig. 2. Architettura logica informale

Rispetto all'architettura risultante dall'analisi dei requisiti sono stati inoltre aggiunti i componenti `Controller` e `SoftwareAgent`.

5.3 Controller

Il ruolo del `Controller` è quello di gestire il modello delle risorse in base ai dati ricevuti dai sensori e alla logica applicativa, in accordo con il pattern MVC: resource model, sensori/attuatori e controller.

Modelliamo quindi il `Controller` come un attore in grado di ricevere dati in input dagli altri componenti del sistema ed emettere in output informazioni sullo stato del modello sotto forma di eventi `ctrlEvent : ctrlEvent(CATEG,NAME,VALUE)`.

```
1 System finalTask2018
2
3 Dispatch cmd : cmd(CMD)
4 Event temperature : temperature(T)
5 Event clock : clock(H,M)
```

```
Event frontSonar : sonar(DISTANCE)
Event sonarSensor : sonar(NAME,DISTANCE)
Event lightCmd : lightCmd(STATE)

Event usercmd : usercmd(CMD)
Event ctrlEvent : ctrlEvent(CATEG,NAME,VALUE)
Event inputCtrlEvent : inputEvent(CATEG,NAME,VALUE)

Context ctxApplPA ip [ host="localhost" port=8018 ] -httpserver
EventHandler evadapterpa for temperature, clock {
       // maps temperature and clock into inputCtrlEvent
       emit inputCtrlEvent fromContent temperature(T) to inputEvent(
           temperature,t1,T);
       emit inputCtrlEvent fromContent clock(H,M) to inputEvent(clock,c1,
           h(H,M))
};
```

Listing 1.10. ../it.unibo.finalTask2018/src/problemAnalysis.qa

Tramite un *EventHandler*, ogni evento emesso dai sensori viene mappato in `inputCtrlEvent : inputEvent(CATEG,NAME,VALUE)` ad uso interno del `Controller`, così che eventi provenienti da sensori fisici diversi possano essere trattati allo stesso modo e non venire mai persi, essendo l'*EventHandler* un componente event-driven.

Per discriminare la natura degli eventi ricevuti, il `Controller` può usare i campi `CATEG` e `NAME` presenti nel payload.

Come già anticipato, ad ogni modifica del Resource Model relativa allo stato di un attuatore, il `Controller` emette a sua volta eventi `ctrlEvent` per notificare il cambiamento ai vari attuatori fisici.

```
QActor controllerpa context ctxApplPA {
Rules{ // The model is in the theory resourceModel.pl
       // se cambia il modello della temperatura
       // R-TempKo: fermo il robot se temperatura >= limite
       changedModelAction(temperature,t1,V) :-
                     limitTemperatureValue(MAX),
                     eval(ge,V,MAX),
                     changeModelItem(robot,r1,h(0)).

       // se cambia il modello del clock
       // R-TimeKo: fermo il robot se il tempo non e' nell'intervallo
       changedModelAction(clock,c1,h(HC,MC)) :-
                     non(inTime(HC,MC)),
                     changeModelItem(robot,r1,h(0)).

       // se cambia il modello del robot
       changedModelAction(robot,R,CMD) :-
```

```
46                      emitevent(ctrlEvent,ctrlEvent(robot,R,CMD)),
47                      fail. // prosegui...
48
49      // R-BlinkLed/R-BlinkHue
50      changedModelAction(robot,R,w(X)) :-
51                      changeModelItem(led,l1,blink).
52      changedModelAction(robot,R,s(X)) :-
53                      changeModelItem(led,l1,blink).
54      changedModelAction(robot,R,h(X)) :-
55                      changeModelItem(led,l1,off).
56
57      // dopo una svolta il modello del robot deve risultare fermo
58      changedModelAction(robot,R,a(X)) :-
59                      changeModelItem(robot,R,h(X)).
60      changedModelAction(robot,R,d(X)) :-
61                      changeModelItem(robot,R,h(X)).
62
63      // se cambia il modello del led
64      changedModelAction(led,L,X) :-
65                      emitevent(ctrlEvent,ctrlEvent(led,L,X)).
66
67      // UTILITIES
68      // vero se il primo tempo e' precedente al secondo
69      timeBefore(H,M1,H,M2) :-
70                      eval(lt,M1,M2), !.
71      timeBefore(H1,_,H2,_) :-
72                      eval(lt,H1,H2).
73
74      // vero se HC,MC e' dentro all'intervallo di tempo consentito
75      inTime(HC,MC) :-
76                      timeInterval(h(H1,M1),h(H2,M2)),
77                      timeBefore(H1,M1,HC,MC),
78                      timeBefore(HC,MC,H2,M2).
79
80      // possibilita' di muovere il robot (R-TempOk e R-TimeOk)
81      canMove :-
82                      limitTemperatureValue(MAX),
83                      getModelItem(sensor,temperature,t1,T),
84                      eval(lt,T,MAX),
85                      getModelItem(sensor,clock,c1,h(HC,MC)),
86                      inTime(HC,MC).
87
88      // R-TempOk e R-TimeOk: cambio il modello solo se il robot puo'
            muoversi
89      changeRobotModel(CMD) :-
90                      canMove,
91                      changeModelItem(robot,r1,CMD).
92  }
93
94      Plan init normal [
```

```
95                demo consult("./resourceModel.pl");
96                println( controllerqa(starts) )
97        ]
98        switchTo waitForInputEvent
99
100       Plan waitForInputEvent [ // debug: print model
101               [ !? getModelItem(r1,VALUE) ] println(model(r1,VALUE));
102               [ !? getModelItem(l1,VALUE) ] println(model(l1,VALUE));
103               [ !? getModelItem(t1,VALUE) ] println(model(t1,VALUE));
104               [ !? getModelItem(c1,VALUE) ] println(model(c1,VALUE));
105               println("--------------------------------------")
106       ]
107       transition stopAfter 3600000
108               whenEvent inputCtrlEvent -> handleInputEvent,
109               whenMsg cmd -> handleCmd
110       finally repeatPlan
111
112       Plan handleInputEvent resumeLastPlan [
113               printCurrentEvent;
114               onEvent inputCtrlEvent : inputEvent(CATEG,NAME,VALUE) ->
115                       demo changeModelItem(CATEG,NAME,VALUE)
116       ]
117
118       Plan handleCmd resumeLastPlan [
119               onMsg cmd : cmd(X) -> println(ricevuto(cmd(X)));
120               onMsg cmd : cmd(X) -> demo changeRobotModel(X)
121       ]
122 }
```

Listing 1.11. ../it.unibo.finalTask2018/src/problemAnalysis.qa

Attualmente sono soddisfatti i requisiti R-TempOk e R-TimeOk, in quanto il `Controller` blocca ogni modifica allo stato del robot al di fuori di un certo intervallo temporale o in presenza di temperature troppo elevate. Il robot viene inoltre fermato non appena queste condizioni non sono più verificate (R-TempKo e R-TimeKo).

Oltre a questo, sono anche soddisfatti i requisiti relativi al lampeggiamento del `Led` e della `HueLamp` (R-BlinkLed e R-BlinkHue), dal momento che alla modifica del modello del robot segue l'emissione di un opportuno evento `ctrlEvent` ad uso degli attuatori della categoria "led", mappato poi da un apposito *EventHandler* nel rispettivo evento `lightCmd`.

```
20 EventHandler evledpa for ctrlEvent { // filter ctrlEvent by CATEGORY and
       NAME
21         emit lightCmd fromContent ctrlEvent(led,l1,CMD) to lightCmd(CMD)
22 };
```

Listing 1.12. ../it.unibo.finalTask2018/src/problemAnalysis.qa

5.4 Software Agent

`SoftwareAgent` realizza la logica applicativa in collaborazione con il `Controller`, occupandosi di manovrare il robot secondo il vincolo R-FloorClean. Decidiamo quindi di modellarlo come un attore nel suo stesso contesto.

L'agent si mette in ascolto degli eventi provenienti dai vari sonar e invia opportuni messaggi `cmd` al controller, gli stessi utilizzati da `HumanOperator` per muovere il robot.

In una fase preliminare, per motivi di testing, `SoftwareAgent` fa semplicemente ruotare il robot quando esso viene rilevato dai sonar a distanza ravvicinata.

```
126 QActor swagpa context ctxApplPA {
127         Rules{
128                 isClose :-
129                         sonarDetect(_,D),
130                         eval(gt,D,0), !,
131                         eval(lt,D,5).
132 
133                 isClose :-
134                         sonarDetect(_,D),
135                         eval(minus,0,D,R),
136                         eval(lt,R,5).
137         }
138 
139         Plan init normal [
140                 println("swag start")
141         ]
142         switchTo waitForSonar
143 
144         Plan waitForSonar [ ]
145         transition stopAfter 3600000
146                 whenEvent frontSonar -> handleFront,
147                 whenEvent sonarSensor -> handleSensor
148         finally repeatPlan
149 
150         Plan handleFront resumeLastPlan [
151                 printCurrentEvent;
152                 onEvent frontSonar : sonar(D) -> forward controllerpa -m
                        cmd : cmd(h(0))
153         ]
154 
155         Plan handleSensor resumeLastPlan [
156                 printCurrentEvent;
157                 onEvent sonarSensor : sonar(N,D) -> addRule sonarDetect(N,D
                        );
158 
159                 // testing: robot close to a sonar makes some moves
```

```
160                 [ !? isClose ] {
161                         forward controllerpa -m cmd : cmd(d(0));
162                         delay 250;
163                         forward controllerpa -m cmd : cmd(d(0));
164                         delay 250;
165                         forward controllerpa -m cmd : cmd(d(0));
166                         delay 250;
167                         forward controllerpa -m cmd : cmd(d(0));
168                         delay 250;
169                         forward controllerpa -m cmd : cmd(s(0));
170                         delay 250;
171                         forward controllerpa -m cmd : cmd(h(0))
172                 };
173                 [ ?? sonarDetect(N,D) ] println(removing(sonarDetect(N,D)))
174         ]
175 }
```

Listing 1.13. ../it.unibo.finalTask2018/src/problemAnalysis.qa

5.5 Changes to HumanOperator

Il sistema deve inoltre fornire un'interfaccia grafica per permettere l'interazione con l'utente da qualsiasi dispositivo (`R-Start`). Ciò può essere realizzato in prima battuta tramite l'inserimento del flag `-httpserver` nel contesto dell'applicazione, sfruttando il server HTTP fornito dall'infrastruttura *QActor*, con l'idea di poterlo sostituire in futuro con uno più sofisticato (ad esempio per realizzare l'autenticazione degli utenti richiesta nei requisiti `R-Start` e `R-Stop`).

```
14 Context ctxApplPA ip [ host="localhost" port=8018 ] -httpserver
```

Listing 1.14. ../it.unibo.finalTask2018/src/problemAnalysis.qa

La GUI web ottenuta tramite il flag `-httpserver` è dotata di bottoni che, quando vengono premuti, scatenano eventi `usercmd : usercmd(robotgui(X))`. `HumanOperator` deve quindi catturare questi eventi e inoltrarli sotto forma di messaggi `cmd : cmd(X)` al `Controller`.

`HumanOperator` è ancora un emettitore di comandi per il robot come visto nella sezione precedente, tuttavia ora è in grado di generarli in base ai pulsanti premuti dall'utilizzatore umano sulla console.

```
179 QActor humanoperatorpa context ctxApplPA {
180         Plan init normal [
181                 println("humanOperator start")
182         ]
183         switchTo waitForUserCmd
184
```

```
185         Plan waitForUserCmd [ ]
186         transition stopAfter 3600000
187                 whenEvent usercmd -> sendCmd
188         finally repeatPlan
189
190         Plan sendCmd resumeLastPlan [
191                 onEvent usercmd : usercmd(robotgui(CMD)) -> println(
                        robotgui(CMD));
192                 onEvent usercmd : usercmd(robotgui(CMD)) -> forward
                        controllerpa -m cmd : cmd(CMD)
193         ]
194 }
```

Listing 1.15. ../it.unibo.finalTask2018/src/problemAnalysis.qa

5.6 Changes to Robot

Il robot si mette in attesa di messaggi `moveRobot` e li interpreta come comandi di movimento, tuttavia i cambiamenti nel modello del robot vengono notificati all'esterno dal controller sotto forma di eventi `ctrlEvent`.

Similmente a quanto fatto nella Subsection 4.9, per effettuare la conversione da `ctrlEvent` a messaggi abbiamo utilizzato un *EventHandler*, così da accodare gli eventi senza il rischio di perderne qualcuno.

```
1  System finalTask2018
2
3  Event frontSonar : sonar(DISTANCE)
4  Event sonarSensor : sonar(NAME,DISTANCE)
5  Dispatch moveRobot : moveRobot(CMD)
6
7  Event ctrlEvent : ctrlEvent(CATEG,NAME,CMD)
8  Event local_robotCmd : moveRobot(CMD)
9
10 Context ctxVirtualRobotNode ip [ host="localhost" port=8822 ]
11 EventHandler evhnode for ctrlEvent {
12         emit local_robotCmd fromContent ctrlEvent(robot,r1,CMD) to
               moveRobot(CMD)
13 };
14 EventHandler evhnodebroker for local_robotCmd {
15         forwardEvent robotnode -m moveRobot  // stesso payload
16 };
17
18 Context ctxApplPA ip [ host="localhost" port=8018 ] -standalone
```

Listing 1.16. ../it.unibo.finalTask2018/src/virtualRobotNode.qa

Poiché gli *EventHandler* possono solo convertire eventi in messaggi con lo stesso payload, introduciamo un ulteriore evento

`local_robotCmd : moveRobot(CMD)` ad uso interno dei due *EventHandler* che effettuano il mapping.

5.6.1 Virtual and Real robots

Concettualmente, il modello del robot virtuale è esattamente lo stesso della Subsection 4.9. L'unica differenza risiede nella diversa implementazione dei metodi Java chiamati con il comando `javaRun`.

Si rimanda questo argomento più nel dettaglio nella Subsection 7.3.

Per quanto riguarda il robot fisico, questi viene al momento simulato tramite un oggetto Java che si limita a stampare a video i comandi ricevuti.

Un attore a parte si occupa invece di gestire lo stato del led, anch'esso rappresentato da un oggetto mock.

```
44 QActor ledrobotmock context ctxRealRobotMock {
45         Plan init normal [
46                 javaRun it.unibo.finalTask2018.adapter.lampAdapter.init()
47         ]
48         transition stopAfter 3600000
49                 whenEvent lightCmd -> handleCmd
50         finally repeatPlan
51
52         Plan handleCmd resumeLastPlan [
53                 onEvent lightCmd : lightCmd(CMD) -> javaRun it.unibo.
                        finalTask2018.adapter.lampAdapter.setLamp(CMD)
54         ]
55 }
```

Listing 1.17. ../it.unibo.finalTask2018/src/realRobotMock.qa

Da notare come entrambi i robot, reagendo agli stessi eventi `ctrlEvent` generati al cambiamento del modello del robot, possono eseguire in contemporanea le medesime mosse.

6 Project

Al momento, l'applicazione è in grado di soddisfare i seguenti requisiti:

- `R-TempOk`, `R-TimeOk`, `R-TempKo` e `R-TimeKo`, ad opera di opportune regole Prolog presenti nella base di conoscenza del `Controller`.
- `R-BlinkLed` e `R-BlinkHue`[4], mediante l'emissione da parte del controller di eventi `ctrlEvent : ctrlEvent(led,l1,CMD)` (poi mappati in `lightCmd`) a seguito della modifica del modello del led in base allo stato di movimento del

[4] Resta da sviluppare l'implementazione della classe Java *technology-dependent* che dovrà interagire in modo RESTful con la `Hue Lamp` remota

robot. L'agent della Hue Lamp e il robot fisico (attore `ledrobotmock`) sono sensibili a questi eventi.

I relativi test vengono trattati nella Subsection 8.2.5.

Rimangono da affrontare i restanti requisiti relativi al movimento autonomo del robot (pilotato da `SoftwareAgent`):

1. `R-Start` e `R-Stop`: la pulizia della stanza inizia e termina preventivamente a seguito di comandi `start/stop` inviati da un *utente autorizzato*.
2. `R-FloorClean` e `R-End`: il robot deve pulire l'intera superficie della stanza (da `sonar1` a `sonar2`), fermandosi al termine del lavoro.
3. `R-AvoidMobile`, `R-AvoidFix` e `R-Obstacle`: il robot deve essere in grado di rilevare ed evitare gli ostacoli sia fissi sia mobili, eventualmente fermandosi nel caso ciò non sia possibile.

6.1 Project architecture

Il ruolo centrale ricoperto dal controller ci suggerisce di impiegarlo come *system integrator* per l'architettura di progetto, sfruttando la propagazione delle informazioni offerta dall'infrastruttura *QActor*. Ogni componente (nella forma di un micro-servizio) infatti, per entrare a far parte del sistema, deve unicamente conoscere il contesto del controller, dal quale potrà ricevere a runtime la conoscenza delle altre entità presenti.

Tra queste, una dovrà offrire un supporto per l'interazione *human-to-machine*: in particolare, occorre consentire l'invio di comandi per il robot da qualsiasi dispositivo e permettere l'autenticazione degli utenti. A tal fine, introduciamo un frontend server raggiungibile via web tramite un'interfaccia RESTful, poiché standard.

Nell'architettura di progetto della Fig. 3 bisogna quindi considerare anche i browser degli utenti (o un qualsiasi altro client RESTful), il server HTTP che gestisce le richieste e una base dati per la persistenza necessaria all'autenticazione.

Una possibile alternativa potrebbe essere sostituire l'infrastruttura *QActor* con un modello del tipo publish/subscribe per la propagazione delle informazioni all'interno del sistema. In questo scenario, ogni entità è svincolata dalle altre, in quanto deve conoscere unicamente il server presso il quale registrarsi come publisher o subscriber, ad esempio un server MQTT.

Considerazioni a questo proposito vengono presentate nella Subsection 7.2.

6.2 Starting and stopping the agent

Prima di passare all'implementazione del frontend (Subsection 7.2), per avere in breve tempo un prototipo funzionante della console, decidiamo di modificare quella fornita tramite il flag `-httpserver`.

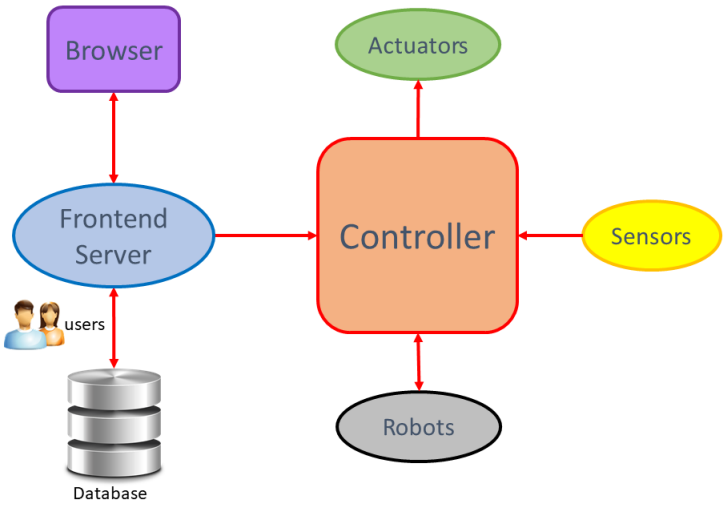


Fig. 3. Architettura di progetto

Questa, oltre ai bottoni che corrispondono ai movimenti elementari del robot, presenta due pulsanti che possono essere utilizzati per l'avvio e la terminazione della pulizia della stanza ad opera di `SoftwareAgent`.

Attualmente la console emette eventi `cmd : cmd(CMD)` alla pressione di questi pulsanti, abbiamo però deciso di sostituirli con eventi `alarm : usercmd(CMD)` così che, avendo lo stesso payload degli altri (quelli legati ai movimenti del robot), la loro gestione risulti semplificata.

Poiché vogliamo che nessun comando vada perso, introduciamo un componente event-driven (*EventHandler*) che effettui il mapping degli eventi `alarm` in messaggi `externalcmd` diretti all'agent ed aventi lo stesso payload dei precedenti.

```
1 EventHandler evhswag for alarm {
2         forwardEvent swag -m externalcmd
3 };
```

Listing 1.18. cleaningRobotSystem.qa

6.2.1 R-Start e R-Stop

Alla pressione del pulsante di start, il robot deve trovarsi vicino a `sonar1` per poter partire con la pulizia, in caso contrario il comando viene ignorato. Se

invece viene ricevuto un messaggio di stop (`usercmd(halt)`) durante il movimento del robot, questi deve fermarsi e l'agent deve ritornare nello stato iniziale.

```
Plan init normal [
        println("swAgent: waiting for start command")
]
transition stopAfter 3600000
        whenMsg externalcmd -> receivedCmd
finally repeatPlan

Plan receivedCmd [
        // R-Start
        onMsg externalcmd : usercmd(clean) -> {
                println("ricevuto clean");
                addRule startCmd
        };
        // R-Stop
        onMsg externalcmd : usercmd(halt) -> println("ricevuto halt")
]
transition
        whenTime 800 -> init
        whenEvent [ ?? startCmd ] sonarSensor -> detectedBySonar
```

Listing 1.19. SoftwareAgent, pt1

Se la posizione corrente è in prossimità di `sonar1`, ci aspettiamo di ricevere da questi un evento `sonarSensor` con una distanza inferiore ad una certa soglia. Nel caso invece nessun evento venga rilevato entro un timeout, assumiamo di non essere allineati con un sonar, e quindi sicuramente di non trovarci nella posizione iniziale.

```
Plan detectedBySonar [
        println("detected by a sonar");
        onEvent sonarSensor : sonar(sonar1,D) -> addRule sonarDetect(
            sonar1,D);

        [ !? isCloseTo(sonar1) ] {
                println("close to sonar1");
                selfMsg swagmsg : cmd(clean) // R-Start
        }
        else
                println("NOT close to sonar1");

        removeRule sonarDetect(sonar1,D)
]
transition
        whenTime 800 -> init
        whenMsg swagmsg -> cleaning
```

Listing 1.20. SoftwareAgent, pt2

La vicinanza ad un sonar viene valutata grazie alle seguenti regole Prolog presenti nella base di conoscenza dell'agent:

```
isCloseTo(S) :-
               sonarDetect(S,D),
               eval(gt,D,0), !,
               eval(lt,D,5).

isCloseTo(S) :-
               sonarDetect(S,D),
               eval(minus,0,D,R),
               eval(lt,R,5).
```

Listing 1.21. SoftwareAgent, Rules - pt1

Per quanto riguarda la terminazione, se durante la pulizia arriva il comando `externalcmd : usercmd(halt)`, l'agent transita prima in `receivedCmd` e successivamente nello stato iniziale, in attesa di nuovi comandi.

6.2.2 R-End

Se durante il suo lavoro il robot viene rilevato in prossimità di `sonar2`, possiamo assumere che abbia terminato la pulizia della stanza e che pertanto debba essere fermato. Ciò è possibile reagendo, nello stato di *cleaning*, agli eventi generati dai sonar e transitando in un nuovo stato (`detectedByFinal`) il cui compito è quello di accertarsi che il robot si trovi effettivamente vicino al secondo sonar, similmente a quanto accade in `detectedBySonar` per `sonar1`.

```
Plan cleaning [
       println("cleaning")
       // do something
]
transition stopAfter 3600000
       whenEvent frontSonar -> handleFront,
       whenEvent sonarSensor -> detectedByFinal,
       whenMsg externalcmd -> receivedCmd
```

Listing 1.22. SoftwareAgent, pt3

```
Plan detectedByFinal [
       println("detected by a sonar");
       onEvent sonarSensor : sonar(sonar2,D) -> addRule sonarDetect(
           sonar2,D);
```

```
4
5       [ !? isCloseTo(sonar2) ] {
6               println("close to sonar2");
7               selfMsg swagmsg : cmd(halt) // R-End
8       }
9       else
10              println("NOT close to sonar2");
11
12      removeRule sonarDetect(sonar2,D)
13  ]
14  transition
15      whenTime 800 -> cleaning
16      whenMsg swagmsg -> init
```

Listing 1.23. SoftwareAgent, pt4

Il comportamento di `SoftwareAgent`, per quanto riguarda il soddisfacimento dei vincoli `R-Start`, `R-Stop` e `R-End`, può essere quindi schematizzato dal diagramma degli stati della Fig. 4.

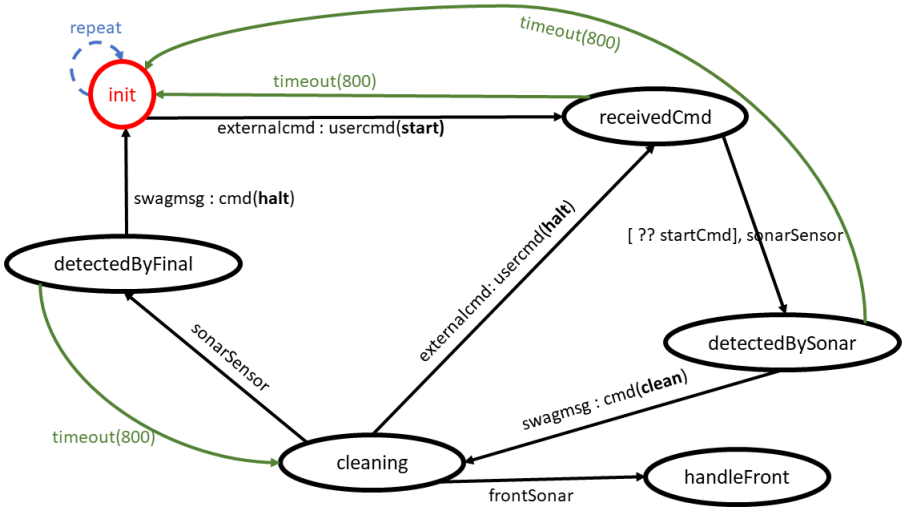


Fig. 4. Diagramma parziale degli stati di `SoftwareAgent` per `R-Start`, `R-Stop` e `R-End`

6.3 Cleaning the room

Durante la sua attività, il robot deve pulire l'intera superficie della stanza (`R-FloorClean`). Come discusso nella Subsection 4.1, l'approccio più semplice per affrontare la pulizia di una stanza rettangolare è quello di partire da un angolo e percorrere a "zig-zag" la stanza in tutta la sua profondità, coprendo progressivamente l'intera larghezza fino a terminare nell'angolo opposto a quello di partenza.

Per quanto riguarda il problema del numero di passi laterali quando la lunghezza della stanza non è un multiplo pari del "passo", nella pratica il robot si sposta lateralmente di meno di un passo quando raggiunge la parete finale (poiché ostacolato da questa), compiendo un'ulteriore striscia così da terminare comunque il conteggio con un numero pari, a discapito di una parziale sovrapposizione della superficie già pulita.

Questa strategia facilita inoltre il testing del requisito: dal momento che la posizione finale corrisponde al punto di arrivo, se il robot viene rilevato in prossimità di `sonar2` è possibile dedurre che abbia pulito l'intera stanza (si veda la Subsection 8.3.2).

I movimenti del robot possono essere quindi riassunti in prima battuta dal diagramma della Fig. 5.

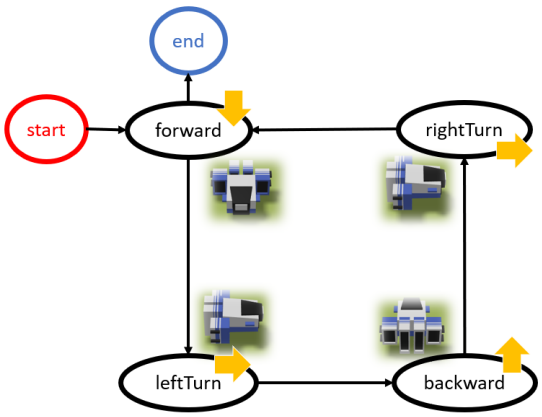


Fig. 5. Schema del movimento del robot durante la pulizia

Partendo da questo modello concettuale di base, introduciamo alcuni stati aggiuntivi che derivano dai vincoli specifici del problema.

In particolare, mentre il raggiungimento della parete frontale (rispetto alla posizione iniziale del robot, ovvero quella opposta a `sonar1`) è segnalato dalla presenza del secondo sonar, per la parete di fondo non vi è altro modo se non

il sonar frontale del robot a rilevare la presenza di un generico ostacolo. Questa problematica verrà ripresa nella sezione successiva.

Un'ulteriore complicazione riguarda la presenza del primo sonar lungo tutta la parete di sinistra, il che ci suggerisce l'inserimento di uno stato aggiuntivo, `startCleaning`, analogo allo stato di `forward`, dove anche in questo caso viene usato eccezionalmente il sonar frontale per gestire la svolta al raggiungimento della parete[5].

```
Plan startCleaning [
        println("start cleaning");
        emit inputCtrlEvent : inputEvent(agent,swag,cleaning); // (1)
        forward controller -m cmd : cmd(w(0))
]
transition stopAfter 3600000
        whenEvent frontSonar -> leftTurn,
        whenEvent externalcmd -> receivedCmd // R-Stop

Plan leftTurn [
        println("left turn");
        forward controller -m cmd : cmd(h(0));
        forward controller -m cmd : cmd(a(0));
        delay 1000;
        forward controller -m cmd : cmd(w(0));
        delay 800;
        forward controller -m cmd : cmd(h(0));
        forward controller -m cmd : cmd(a(0));
        delay 800
]
switchTo backCleaning

Plan backCleaning [
        println("cleaning back");
        forward controller -m cmd : cmd(w(0))
]
transition stopAfter 3600000
        whenEvent frontSonar -> rightTurn,
        whenEvent externalcmd -> receivedCmd // R-Stop

Plan rightTurn [
        println("right turn");
        forward controller -m cmd : cmd(h(0));
        forward controller -m cmd : cmd(d(0));
        delay 1000;
        forward controller -m cmd : cmd(w(0));
        delay 800;
        forward controller -m cmd : cmd(h(0));
        forward controller -m cmd : cmd(d(0));
```

[5] Assumiamo, per semplicità, che non vi siano ostacoli lungo le pareti

```
40          delay 800
41  ]
42  switchTo forwardCleaning
43
44  Plan forwardCleaning [
45          println("cleaning forward");
46          forward controller -m cmd : cmd(w(0))
47  ]
48  transition stopAfter 3600000
49          // la parete di fondo e' segnalata da sonar2
50          whenEvent sonarSensor -> detectedByFinal, // R-End
51          whenEvent externalcmd -> receivedCmd // R-Stop
```

Listing 1.24. SoftwareAgent, pt5

Dallo stato `detectedByFinal`, riportato precedentemente, l'agent transita poi in `leftTurn`, riprendendo il ciclo, oppure nello stato finale `end` a seconda della vicinanza dal secondo sonar.

```
1  Plan end [
2          println("end");
3          emit inputCtrlEvent : inputEvent(agent,swag,idle); // (2)
4          forward controller -m cmd : cmd(h(0))
5  ]
6  switchTo init
```

Listing 1.25. SoftwareAgent, pt6

Quando il robot viene pilotato da un utente umano, in caso di collisione è il controller a rilevare l'evento `frontSonar` e a fermare il robot. Ciò non deve accadere quando è invece il `SoftwareAgent` a controllarlo: per questo motivo l'agent si fa carico di inviare un `inputCtrlEvent` per segnalare al controller quando sta pilotando il robot, così che questi non interferisca (righe (1) e (2)).

Alla ricezione di questi eventi, il controller modifica il modello delle risorse in cui è mantenuto lo stato dell'agent.

```
1  % modello dell'agent
2  model( type(software, agent), name(swag), value(idle) ).
```

Listing 1.26. resourceModel.pl

```
1  Plan handleFront resumeLastPlan [
2          [ !? getModelItem(software,agent,swag,idle) ]
3                  demo changeRobotModel(h(0))
4  ]
```

Listing 1.27. Gestione di `frontSonar` da parte del `Controller`

Il diagramma degli stati risultante è quindi rappresentato nella Fig. 6, dove gli stati rappresentati vanno a sostituire il precedente generico `cleaning`.

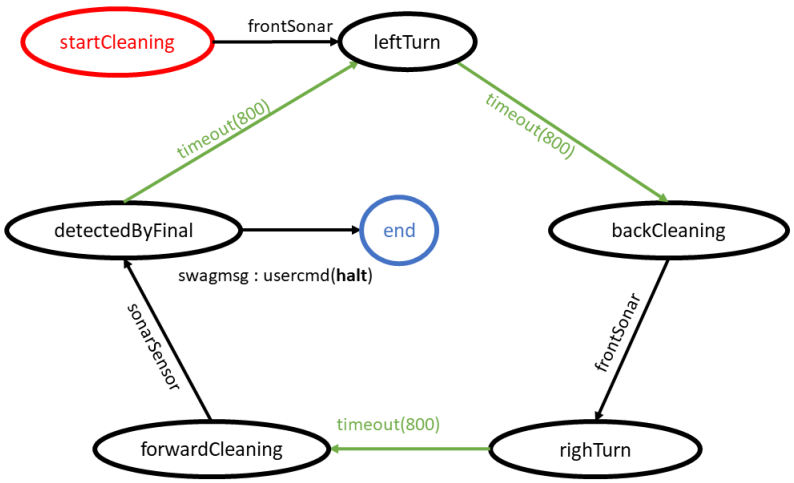


Fig. 6. Diagramma parziale degli stati di `SoftwareAgent` per `R-FloorClean` e `R-End`

6.4 Obstacle avoidance

Gli eventi `frontSonar` permettono a `SoftwareAgent` di individuare gli ostacoli sul cammino del robot. Poiché il robot ha solo un sonar frontale, non è in grado di riconoscere immediatamente che tipo di oggetto ha davanti: se si tratta di una parete o di un ostacolo fisso o mobile da aggirare. In presenza di uno di questi eventi, l'agent deve quindi innanzitutto cercare di distinguere tra pareti ed ostacoli e, in tal caso, tentare di evitarli.

Poiché abbiamo deciso di affrontare i requisiti legati al movimento autonomo del robot in maniera incrementale, in questa sezione ci concentreremo unicamente sulle politiche che regolano il superamento degli ostacoli. Successivamente queste verranno integrate con la logica di pulizia della stanza.

Inizialmente si assume che l'ostacolo sia mobile, si aspetta dunque un piccolo intervallo di tempo per poi riprovare ad avanzare. Se non si ricevono altri eventi dal sonar frontale, si presume di essere riusciti a superare l'ostacolo.

```
Plan handleFront [
       println("handleFront");
       forward controller -m cmd : cmd(h(0));
       println("possible mobile obstacle");
       delay 1000;
       forward controller -m cmd : cmd(w(0));
       println("provo ad avanzare")
]
transition
       whenTime 800 -> avoidMobile
       whenMsg externalcmd -> receivedCmd,
       whenEvent frontSonar -> avoidFix // R-AvoidFix

 // R-AvoidMobile
 Plan avoidMobile [
       println("avoidMobile");
       println("ok, ostacolo superato")
]
switchTo cleaning
```

Listing 1.28. SoftwareAgent, pt7

Viceversa, se si ricevono altri eventi `frontSonar`, l'ostacolo viene considerato fisso e occorre provare ad aggirarlo prima da una direzione e, in caso di fallimento, dall'altra, cercando periodicamente la presenza di passaggi.

```
Plan avoidFix [
       println("avoidFix");
       forward controller -m cmd : cmd(h(0));
       [ !? exploring(DIR) ] println(exploring(DIR))
       else addRule exploring(r);
       // aumento contatore di tentativi
       demo avoidFixTry;
       [ !? avoidFixGiveUp ] {
              [ !? exploring(l) ] selfMsg swagmsg : cmd(giveUpLeft) // R-
                  Obstacle
              else selfMsg swagmsg : cmd(giveUpRight);
              demo switchExplorationDir;
              println("Raggiunti max tentativi")
       } else {
              delay 800;
              println("proviamo a girarci intorno");
              [ !? exploring(l) ] {
                     forward controller -m cmd : cmd(a(0));
                     println("da sinistra")
              }
              else {
                     forward controller -m cmd : cmd(d(0));
                     println("da destra")
```

```
                };
                delay 2000;
                forward controller -m cmd : cmd(w(0));
                delay 1000; // passo laterale
                forward controller -m cmd : cmd(h(0));
                delay 2000
        }
]
transition
        whenTime 800 -> checkDoor
        whenEvent frontSonar -> failure,
        whenMsg externalcmd -> receivedCmd,
        whenMsg swagmsg -> giveUp // esaminare il payload

Plan checkDoor [
        println("checkDoor");
        [ !? exploring(r) ] forward controller -m cmd : cmd(a(0))
        else forward controller -m cmd : cmd(d(0));
        delay 800;
        forward controller -m cmd : cmd(w(0));
        delay 800; // testata
        forward controller -m cmd : cmd(h(0));
        delay 2000
]
transition
        whenTime 800 -> doorFound
        whenMsg externalcmd -> receivedCmd,
        whenEvent frontSonar -> avoidFix

Plan doorFound [
        println("doorFound");
        forward controller -m cmd : cmd(w(0));
        delay 1000; // supero l'ostacolo
        forward controller -m cmd : cmd(h(0));
        [ !? exploring(r) ] forward controller -m cmd : cmd(a(0))
        else forward controller -m cmd : cmd(d(0))
]
switchTo goToPrevLevel
```

Listing 1.29. SoftwareAgent, pt8

Il fallimento è determinato dalla presenza di un nuovo ostacolo che impedisce di proseguire oltre in quella direzione o dopo un certo numero di tentativi non riusciti. Se si fallisce in entrambe le direzioni, la pulizia termina (R-Obstacle).

```
Plan failure [
        println("failure");
        [ !? exploring(r) ] {
                println("failure da destra");
```

```
                selfMsg swagmsg : cmd(giveUpRight);
                delay 800;
                forward controller -m cmd : cmd(a(0))
        }
        else {
                println("failure da sinistra");
                selfMsg swagmsg : cmd(giveUpLeft)
        };
        demo switchExplorationDir
]
transition stopAfter 3600000
        whenMsg swagmsg -> giveUp

// ho raggiunto max tentativi
Plan giveUp [
        println("giveUp");
        // elimino l'ultimo incremento, quello che ha fatto raggiungere la
             soglia
        demo decremFoundFix;
        onMsg swagmsg : cmd(giveUpRight) -> {
                delay 800;
                forward controller -m cmd : cmd(a(0));
                delay 800;
                selfMsg swagmsg : cmd(resumeLastPos)
        };
        onMsg swagmsg : cmd(giveUpLeft) -> { // R-Obstacle
                removeRule foundFix(X);
                removeRule exploring(l)
        }
]
transition
        whenTime 800 -> end
        whenMsg swagmsg -> resumeLastPosition

Plan resumeLastPosition [
        println("resume last position");
        delay 2000;
        [ !? counter(foundFix,C) ] {
                demo decremFoundFix;
                forward controller -m cmd : cmd(w(0));
                delay 800; // passo
                forward controller -m cmd : cmd(h(0));
                delay 2000
        }
        else {
                selfMsg swagmsg : cmd(initialP); // raggiunta posizione di
                    partenza
                forward controller -m cmd : cmd(d(0))
        }
]
```

```
transition
        whenTime 800 -> resumeLastPosition
        whenMsg externalcmd -> receivedCmd,
        whenMsg swagmsg -> avoidFix // prova dall'altra parte
```

Listing 1.30. SoftwareAgent, pt9

Se invece viene trovato un passaggio, dopo averlo superato il robot deve ritornare alla stessa altezza a cui si trovava prima dell'ostacolo.

```
Plan goToPrevLevel [
        println("goToPrevLevel");
        delay 2000;
        [ !? counter(foundFix,C) ] {
                demo decremFoundFix;
                forward controller -m cmd : cmd(w(0));
                delay 800; // passo
                forward controller -m cmd : cmd(h(0))
        }
        else {
                selfMsg swagmsg : cmd(initialP); // raggiunta posizione di
                    partenza
                [ !? exploring(r) ] forward controller -m cmd : cmd(d(0))
                else forward controller -m cmd : cmd(a(0));
                delay 800;
                println("riprendo la direzione di marcia")
        }
]
transition
        whenTime 300 -> goToPrevLevel
        whenMsg externalcmd -> receivedCmd,
        whenMsg swagmsg -> cleaning
```

Listing 1.31. SoftwareAgent, pt10

Gli stati che implementano le politiche di *obstacle avoidance* possono essere riassunti con la Fig. 7.

Per gestire il passaggio fra i suoi stati interni, l'agent utilizza dei messaggi "privati" `swagmsg : cmd(CMD)` diretti a se stesso, mentre le due fasi di esplorazione, a destra e a sinistra dell'ostacolo, vengono realizzate dagli stessi stati discriminando i due casi tramite la regola `exploring(DIR)`.

```
switchExplorationDir :-
        exploring(r),
        retract(exploring(r)),
        assert(exploring(l)).
```

Listing 1.32. SoftwareAgent, Rules - pt2

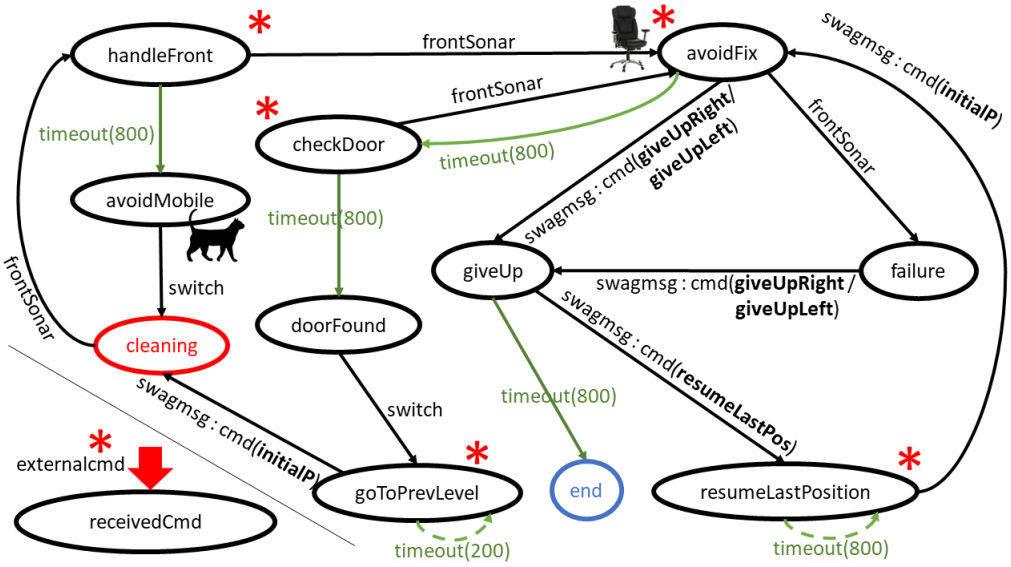


Fig. 7. Diagramma parziale degli stati di `SoftwareAgent` per `R-AvoidFix`, `R-AvoidMobile` e `R-Obstacle`

Per realizzare il comportamento a tentativi del robot vengono impiegate le seguenti regole Prolog che implementano un contatore.

```
giveUpLimit(3).

increment(C) :-
        counter(C,N), !,
        N2 is N+1,
        retract(counter(C,N)),
        assert(counter(C,N2)).

increment(C) :-
        assert(counter(C,1)).

increment(C,1) :- !,
        increment(C).

increment(C,N) :-
        increment(C),
        eval(minus,N,1,N2),
        increment(C,N2).

decrement(C) :-
        counter(C,1), !,
```

```
22        retract(counter(C,1)).
23
24 decrement(C) :-
25        counter(C,N), !,
26        eval(minus,N,1,N2),
27        retract(counter(C,N)),
28        assert(counter(C,N2)).
29
30 avoidFixTry :-
31        increment(foundFix).
32
33 avoidFixGiveUp :-
34        giveUpLimit(L),
35        counter(foundFix,L).
36
37 decremFoundFix :-
38        decrement(foundFix).
```

Listing 1.33. SoftwareAgent, Rules - pt3

6.5 Complete Agent

Per integrare l'*obstacle avoidance* con la pulizia della stanza sono necessari alcuni accorgimenti.

Poiché abbiamo assunto che non vi siano ostacoli lungo le pareti, gli unici stati sensibili agli eventi del sonar frontale sono `forwardCleaning` e `backCleaning`, ai quali va quindi aggiunta la transizione che originariamente si trovava nello stato fittizio `cleaning`.

Inoltre, dopo aver aggirato un ostacolo, l'agent deve tornare nello stato "di pulizia" (forward o backward) in cui si trovava in precedenza: a tal fine abbiamo introdotto lo stato `selectDirection`, che tramite opportune regole discrimina lo stato futuro in cui deve portarsi. Tali regole devono essere asserite nei rispettivi stati di provenienza.

```
1  Plan selectDirection [
2         println("selectDirection");
3         removeRule exploring(X);
4         [ !? direction(f) ] println("f");
5         [ !? direction(b) ] { println("b"); selfMsg swagmsg : cmd(b) }
6  ]
7  transition
8         whenTime 800 -> forwardCleaning
9         whenMsg swagmsg -> backCleaning
10
11 Plan forwardCleaning [
12        println("cleaning forward");
```

```
13        [ !? direction(D) ]
14                println(direction(D)) // debug
15        else
16                addRule direction(f);
17        forward controller -m cmd : cmd(w(0))
18 ]
19 transition stopAfter 3600000
20        whenEvent sonarSensor -> detectedByFinal, // R-End
21        whenEvent frontSonar -> handleFront,
22        whenEvent externalcmd -> receivedCmd // R-Stop
23
24 Plan backCleaning [
25        println("cleaning back");
26        [ !? direction(D) ]
27                println(direction(D)) // debug
28        else
29                addRule direction(b);
30        forward controller -m cmd : cmd(w(0))
31 ]
32 transition stopAfter 3600000
33        whenEvent frontSonar -> frontSonarDetected,
34        whenEvent externalcmd -> receivedCmd // R-Stop
```

Listing 1.34. SoftwareAgent, pt11

Per distinguere la parete di fondo da un ostacolo, entrambi segnalati da un evento `frontSonar`, utilizziamo un contatore per misurare la profondità della stanza durante il primo tratto, corrispondente a `startCleaning`, e uno durante ogni pulizia "backward". In questo modo, alla ricezione dell'evento, è possibile capire se si debba tornare indietro (`rightTurn`) o evitare l'ostacolo.

```
1  Plan startCleaning [
2         // [...]
3  ]
4  switchTo countRoomLen
5
6  Plan countRoomLen [
7         demo increment(roomLen) // counter(roomLen,C++)
8  ]
9  transition
10        whenTime 300 -> countRoomLen
11        whenEvent frontSonar -> leftTurn,
12        whenEvent externalcmd -> receivedCmd // R-Stop
13
14 Plan backCleaning [
15        // [...]
16 ]
17 switchTo countSteps
18
```

```
Plan countSteps [
        demo increment(steps) // counter(steps,C++)
]
transition
        whenTime 300 -> countSteps
        whenEvent frontSonar -> frontSonarDetected,
        whenEvent externalcmd -> receivedCmd // R-Stop

Plan frontSonarDetected [
        println("frontSonarDetected");
        [ !? counter(roomLen,C)] println(roomLen(C));
        [ !? counter(steps,N)] println(steps(N));

        [ !? isInWallProximity ] {
                println("parete");
                [ !? direction(b) ] selfMsg swagmsg : cmd(r)
        }
        else
                println("ostacolo")
]
transition
        whenTime 800 -> handleFront // obstacle avoidance
        whenMsg swagmsg -> rightTurn // wall
```

Listing 1.35. SoftwareAgent, pt12

Se i passi contati sono maggiori o minori della dimensione della stanza, – entro una certa soglia, per tolleranza ad eventuali errori nel conteggio – allora il robot si trova davanti alla parete di fondo.

```
isInWallProximity :-
        counter(steps,N),
        counter(roomLen,M),
        eval(minus,M,N,R),
        eval(lt,R,4).
```

Listing 1.36. SoftwareAgent, Rules - pt4

Per quanto riguarda il raggiungimento della parete frontale, segnalata dalla presenza di `sonar2`, occorre tenere in considerazione il fatto che il robot potrebbe comunque urtare la parete, generando un evento `frontSonar`. Se tale evento viene ricevuto quando l'agent si trova già in `backCleaning`, questi potrebbe scambiarlo erroneamente per un ostacolo. Pertanto, abbiamo inserito uno stato aggiuntivo per catturare l'eventuale `frontSonar` e "consumarlo" prima di transitare in uno stato sensibile agli eventi del sonar frontale.

```
Plan leftTurn [
        // [...]
```

```
]
transition
        whenTime 800 -> waitForGodot

// "consumo" eventuali eventi frontSonar residui...
Plan waitForGodot [
        println("waiting for Godot...")
]
transition
        whenTime 2000 -> backCleaning // tutto ok, procedi
        whenEvent frontSonar -> backCleaning // evento "consumato"
```

Listing 1.37. SoftwareAgent, pt13

7 Implementation

7.1 Hue Lamp interaction

L'interazione con la Hue Lamp Philips avviene mediante un'interfaccia di API RESTful[6] fornita da un apposito bridge.

Per poter utilizzare la lampada è quindi necessario individuare prima il bridge nella rete ed ottenere uno username. Entrambe le operazioni possono essere facilmente svolte con messaggi HTTP: la prima attraverso una GET ad un preciso URL (`https://discovery.meethue.com/`), mentre la seconda grazie alle API del bridge precedentemente individuato.

Attualmente l'`HueLampAgent` presente nel nostro sistema si rivolge tramite `javaRun` ad un oggetto Java che simula la lampada Wi-Fi; ci viene quindi facile sostituirne l'implementazione con un'altra classe Java che funga da adapter verso le API del bridge della Hue Lamp.

```
        public static void setUp(QActor qa) throws ClientProtocolException
            , IOException {
               File f = new File(FILE_NAME);
               if (readSettingsFile(f)) { // controllo esistenza file (e
                    ben formato: 3 righe)
                  if (checkBridge()) { // controllo bridge
                         if(!checkLamp()) { // lamp non valida, ne
                              cerco un'altra
                            findLamp();
                            updateSettingsFile(f);
                         }
                  } else { // bridge non valido, cerco ip bridge ed
                       id lamp
```

[6] https://it.wikipedia.org/wiki/Representational_State_Transfer

```
38                              findBridge();
39                              getUsername();
40                              findLamp();
41                              updateSettingsFile(f);
42                      }
43              } else { // file non trovato, cerco ip bridge ed id lamp
44                      f.createNewFile();
45                      findBridge();
46                      getUsername();
47                      findLamp();
48                      updateSettingsFile(f);
49              }
50      }
51
52      public static void setLampState(QActor qa, String state) throws
            ClientProtocolException, IOException {
53              JSONObject obj = new JSONObject();
54              HttpResponse response = null;
55              if (state.equals("on")) {
56                      obj.accumulate("on", true);
57                      obj.accumulate("effect", "none");
58                      response = RESTfulClient.execPUT("http://" + bridge
                             + "/api/" + username + "/lights/" + lamp + "/
                            state",
59                                      obj.toString());
60              } else if (state.equals("off")) {
61                      obj.accumulate("on", false);
62                      obj.accumulate("effect", "none");
63                      response = RESTfulClient.execPUT("http://" + bridge
                             + "/api/" + username + "/lights/" + lamp + "/
                            state",
64                                      obj.toString());
65              } else if (state.equals("blink")) {
66                      obj.accumulate("on", true);
67                      obj.accumulate("effect", "colorloop"); // ->
                            multicolor loop :)
68                      response = RESTfulClient.execPUT("http://" + bridge
                             + "/api/" + username + "/lights/" + lamp + "/
                            state",
69                                      obj.toString());
70              }
71              JSONParser.parseJSONArray(response.getEntity().getContent()
                    );
72      }
```

Listing 1.38. ../it.unibo.finaltask2018/src/it/unibo/finaltask2018/adapter/hueLampAdapter.java

Tale classe espone i metodi statici `setUp()` e `setLamp(String state)`: il primo esegue le operazioni per stabilire una connessione col bridge ed ottenere le credenziali, mentre il secondo permette di cambiare lo stato della lampadina

(on/off/blink).

Dal momento che tutte le comunicazioni con il bridge e la lampada avvengono in modo RESTful, abbiamo implementato una classe Java di utilità[7] che, sfruttando un'apposita libreria, semplifichi tali interazioni. Anche per quanto riguarda la scrittura/lettura dei dati in JSON contenuti nel body dei messaggi ci siamo appoggiati ad una libreria.

7.2 Frontend Server

Fin'ora per permettere l'interazione fra utente e robot abbiamo utilizzato il flag `-httpserver` offerto dal linguaggio *QActor*. Tale scelta è stata fatta però in una fase prototipale per questioni di rapidità.

Stando ai requisiti `R-Start` e `R-Stop`, solo gli utenti autorizzati possono accedere alla console, quindi, per una maggiore personalizzazione dell'interfaccia e per gestire l'autenticazione, abbiamo costruito un nuovo *frontend server*.

Per farlo abbiamo utilizzato *Node.js*, perché permette di scrivere in *Javascript* applicazioni web veloci e scalabili lato server: in particolare il suo framework *Express* fornisce dei template per la generazione di pagine web, utility per il routing, ecc.

Per quanto riguarda l'autenticazione degli utenti, la tecnologia impiegata è *MongoDB*, un DBMS non relazionale. Questa scelta è stata fatta per l'ottima integrazione tra Mongo e Node.js: esiste infatti un'estensione per gestire i database Mongo, chiamata *Mongoose*, proprio in Node.

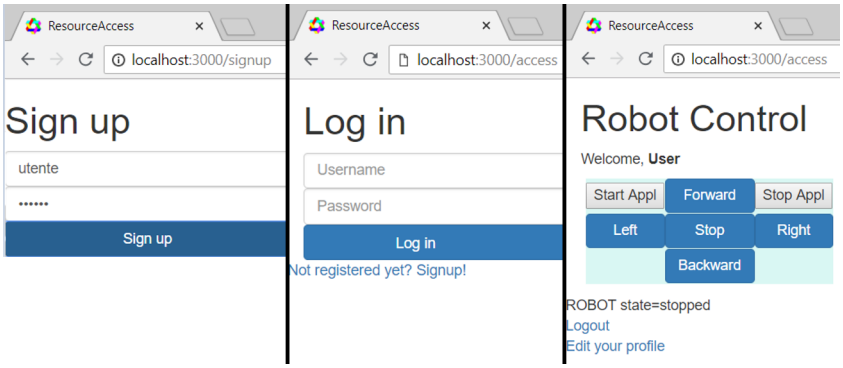


Fig. 8. Registrazione e autenticazione di un utente

[7] RESTfulClient.java

Il frontend server utilizza un proprio modello interno del robot, scritto in JSON, che descrive lo stato in cui si trova e che viene aggiornato in base ai comandi inviati dalla console. Questo modello è svincolato da quello gestito dal controller, pertanto sono presenti tutte le problematiche relative alla sincronizzazione e alla consistenza dei due modelli. Di queste ci occuperemo nella Subsection 7.2.1.

Per interagire con il resto del sistema, il frontend emette eventi `usercmd : usercmd(robotgui(CMD))` alla pressione dei tasti di movimento ed eventi `alarm : usercmd(CMD))` per l'avvio e l'interruzione della pulizia, gli stessi emessi dalla precedente GUI ottenuta con il flag `-httpserver`.

Per la propagazione degli eventi nel sistema è possibile aprire una connessione TCP con il contesto principale dell'applicazione ed inviare su di essa gli eventi nel formato di basso livello utilizzato dall'infrastruttura *QActor*.

Un'alternativa è utilizzare un server Mqtt tramite il quale comunicare messaggi secondo il modello publish-subscribe. In questo caso, lato *QActor*, deve essere presente l'istruzione `pubSubServer` che specifica l'indirizzo e la porta del server Mqtt a cui collegarsi, inoltre tutti gli attori interessati a ricevere o pubblicare informazioni devono essere provvisti del flag `-pubsub`.

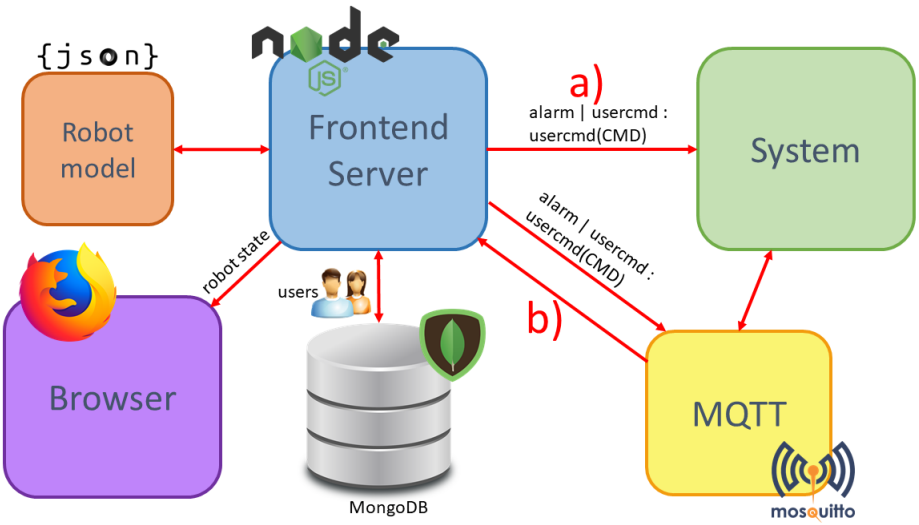


Fig. 9. Schema delle interazioni del frontend server

Per quanto siano attuabili entrambe le soluzioni, la prima non necessita di servizi esterni e permette di passare da un frontend all'altro senza bisogno di alcuna modifica al codice del modello; d'altro canto la seconda evita che i vari

micro-servizi si "conoscano" tra di loro, in quanto ogni componente, per far parte del sistema, deve unicamente dialogare con il server Mqtt.

Il passaggio da una all'altra è comunque facilmente realizzabile settando il parametro `externalActuatorMqtt` usato all'interno del frontend[8].

```
app.post("/robot/actions/commands/h", function(req, res) {
        if(externalActuatorMqtt || externalActuatorSocket)
                delegate("usercmd","usercmd(robotgui( h(low) ))", "stopped"
                    , req, res);
        else
                actuate(‘{ "type": "alarm", "arg": 1000 }‘, "stopped", req,
                     res);
});

var msgNum=1; //parte da 1 e aumentera' via via
function delegate(eventName, payload, newState, req, res) {
        var msg = "msg("+eventName+",event,js,none,"+payload+","+msgNum++
            +")";
        robotModel.robot.state = newState; // (1)
        console.log("emits -> "+ msg);
        try{
                if(externalActuatorMqtt)
                        mqttUtils.publish(msg); //topic = "unibo/qasys"
                else if(externalActuatorSocket)
                        conn.write(msg+"\n");
        }
        catch(e){
                console.log("ERROR "+e);
        }
        res.render("access");
}
```

Listing 1.39. nodeCode/frontend/applCodeRobot.js

7.2.1 Frontend as an observer of the resource model

Alla pressione di un pulsante sulla console non corrisponde sempre un'effettiva modifica del modello del robot, ad esempio quando le condizioni di temperatura e orario non sono soddisfatte, o quando il robot si trova davanti ad un ostacolo; pertanto l'aggiornamento del modello in JSON non deve seguire direttamente all'invio di un comando[9], ma deve essere subordinato alla modifica del modello delle risorse.

Per sincronizzare il modello interno del frontend con quello mantenuto dal controller, il primo deve essere notificato ad ogni cambiamento del secondo. In

[8] Definito nel file di configurazione `settings.json`

[9] Vedi riga con commento `(1)`

altre parole il frontend deve essere un observer del modello delle risorse, le cui modifiche vengono trasmesse all'esterno nella forma di eventi `ctrlEvent`.

Definiamo pertanto un QActor sensibile a tali eventi, in particolare a quelli che riguardano lo stato del robot (similmente a quanto fatto per i robot virtuale e reale), che trasmetta l'informazione aggiornata al server via messaggi HTTP di tipo PUT.

```
System frontendActivator

Event ctrlEvent : ctrlEvent(CATEG,NAME,CMD)
Event local_robotCmd : moveRobot(CMD)
Dispatch moveRobot : moveRobot(CMD)

Context ctxFrontendActivator ip [ host="localhost" port=3434 ]
EventHandler evhfe for ctrlEvent {
        emit local_robotCmd fromContent ctrlEvent(robot,r1,CMD) to
            moveRobot(CMD)
};
EventHandler evhfebroker for local_robotCmd {
        forwardEvent febroker -m moveRobot  // stesso payload
};

Context ctxAppl ip [ host="localhost" port=8018 ] -standalone

QActor feactivator context ctxFrontendActivator {
        Plan init normal [
                println("Activating frontend...");
                // -o shows the output of the Node program
                nodeOp "../nodeCode/frontend/robotFrontendServer.js" -o
        ]
}

QActor febroker context ctxFrontendActivator {
        Plan waitForMsg normal [ ]
        transition stopAfter 3600000
                whenMsg moveRobot : moveRobot(CMD) do
                        javaRun it.unibo.finalTask2018.clientREST.sendPut("
                            http://localhost:3000/robot/r1/state","state",
                            CMD)
        finally repeatPlan
}
```

Listing 1.40. ../it.unibo.finaltask2018/src/frontendActivator.qa

Si noti come nello stesso contesto è presente anche un attore in grado di attivare l'applicazione Node del frontend.

Lato server, basta aggiungere una regola di routing che si occupi di modificare il modello del robot ad ogni richiesta PUT all'apposito indirizzo; lo stato viene poi periodicamente inviato al browser dell'utente tramite una socket.

```
robotStateMap = {h: "stopped", w: "forward", s: "backward", a: "left", d:
     "right"};
app.put("/robot/r1/state", function(req, res) {
       var cmd = req.body.state;
       robotModel.robot.state = robotStateMap[cmd.substr(0,1)];
       console.info("Robot state = " + robotModel.robot.state);
       // the info is sent periodically to the browser by
           showResourceState() in robotFrontendServer.js
       res.sendStatus(200); // OK
});
```

Listing 1.41. nodeCode/frontend/applCodeRobot.js

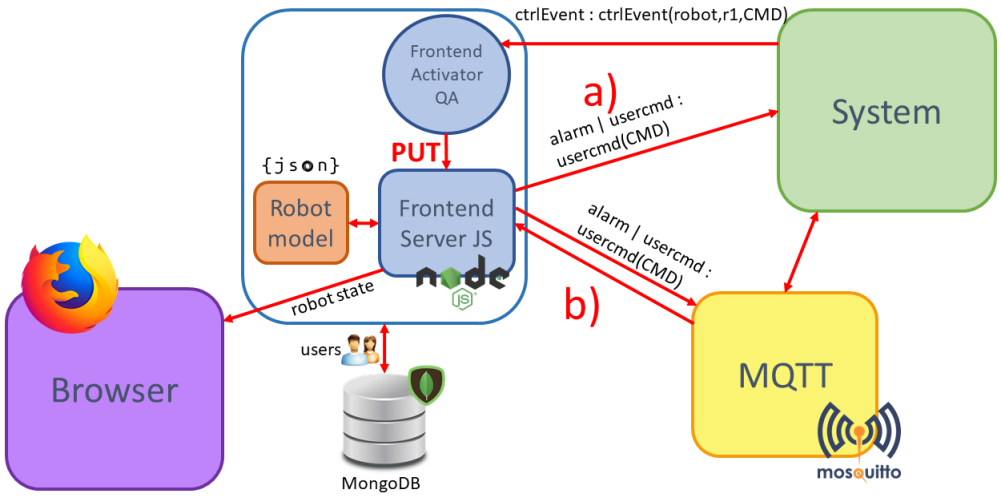


Fig. 10. Refactoring dell'architettura del frontend

7.3 Virtual Robot with Node.js

L'implementazione del robot virtuale si basa sul progetto `configurable-threejs-app`[10] di Pierfrancesco Soffritti, il quale fa uso di

[10] `https://github.com/PierfrancescoSoffritti/configurable-threejs-app`

un server Node.js che può essere raggiunto sia tramite browser, sia tramite una semplice connessione TCP.

Come prima cosa quindi, il robot deve occuparsi di avviare il server e stabilire con questi una connessione.

```
21 QActor robotnode context ctxVirtualRobotNode {
22         Rules {
23                 robotConfig("localhost", 8999). // Node server
24         }
25
26         Plan init normal [
27                 println("robotnode start");
28                 javaRun it.unibo.finalTask2018.adapter.robotAdapter.useImpl
                       ("NodeRobot");
29
30                 [ !? robotConfig(ADDR,PORT) ]
31                         javaRun it.unibo.finalTask2018.adapter.robotAdapter
                               .setUpEnvironment(ADDR,PORT)
32                 else
33                         javaRun it.unibo.finalTask2018.adapter.robotAdapter
                               .setUpEnvironment()
34         ]
35         switchTo waitForCmd
```

Listing 1.42. ../it.unibo.finaltask2018/src/virtualRobotNode.qa

La classe Java fa uso di un file `.bat` per eseguire il server in un nuovo terminale, appositamente aperto, evitando così che questa operazione debba essere svolta manualmente dall'utente, ma allo stesso tempo lasciando all'utilizzatore il controllo per poterlo terminare.

```
11         public void setUpEnvironment(QActor qa, String host, int port) {
12                 System.out.println("setting up the environment...");
13
14                 /* startServerPath.bat
15                  * ---------------------------
16                  * if NOT "%1" == "" cd %1
17                  * cd server
18                  * cd src
19                  * start cmd /k node main 8999
20                  */
21                 try {
22                         Runtime.getRuntime().exec(PATH + "/startServerPath.
                               bat " + PATH); // + " > test.txt 2>&1");
23                         // wait for the server to start...
24                         Thread.sleep(10000); // 10s
25                         clientTcp.initClientConn(qa, host, port);
26                 } catch (Exception e) {
```

```
27                      e.printStackTrace();
28                      throw new RuntimeException(e);
29                  }
30          }
```

Listing 1.43. ../it.unibo.finaltask2018/src/it/unibo/finaltask2018/robot/NodeRobot.java

L'implementazione dei metodi che corrispondono alle varie mosse fa uso della classe di utilità `it.unibo.utils.clientTcp`, la quale costruisce i messaggi nel giusto formato accettato dal server e li invia sulla connessione TCP.

Da questa inoltre, un thread dedicato legge continuamente i dati in arrivo, corrispondenti alle informazioni generate dai sonar fissi e mobili, per poi emettere i relativi eventi `sonarSensor` e `frontSonar`.

7.4 Real Robot with RaspberryPi + Arduino

Il robot fisico che si trova sul RaspberryPi[11] interagisce con il resto del sistema tramite l'invio e la ricezione di eventi (propagati dall'infrastruttura *QActor* via rete) e con Arduino – dove sono collegati sensori e attuatori – tramite una connessione seriale.

L'architettura del sistema può essere quindi riassunta dalla Fig. 11.

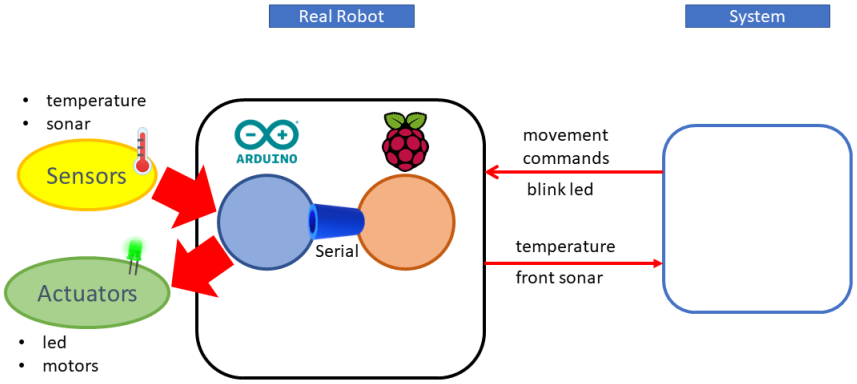


Fig. 11. Schema generale del robot fisico

[11] progetto: `it.unibo.raspRobot`

In realtà, il robot non comunica direttamente con Arduino, in quanto abbiamo scelto di introdurre un ulteriore strato software che esponga una visione di alto livello dei dispositivi fisici collegati alla board di Arduino: questi sono presentati come servizi accessibili via rete (TCP) e gestiti da un opportuno server Python, il quale è l'unico a leggere e scrivere dati sulla seriale (lato Raspberry).

Questa soluzione scarica dal robot fisico la responsabilità di conoscere i dettagli della comunicazione con Arduino, a favore di un'interazione tra entità di alto livello (micro-servizi).

```
1  System finalTask2018
2
3  Event frontSonar : sonar(DISTANCE)    // out
4  Event lightCmd : lightCmd(STATE)    // in
5  Event ctrlEvent : ctrlEvent(CATEG,NAME,CMD)  // in
6  Event local_robotCmd : moveRobot(CMD)   // internal
7  Dispatch moveRobot : moveRobot(CMD)    // internal
8
9  Context ctxRealRobotRasp ip [ host="192.168.43.14" port=8888 ]
10 EventHandler evhreal for ctrlEvent {
11        emit local_robotCmd fromContent ctrlEvent(robot,r1,CMD) to
              moveRobot(CMD)
12 };
13 EventHandler evhrealbroker for local_robotCmd {
14        forwardEvent realrobotrasp -m moveRobot  // stesso payload
15 };
16
17 Context ctxAppl ip [ host="192.168.43.3" port=8018 ] -standalone
18
19 QActor realrobotrasp context ctxRealRobotRasp {
20        Plan init normal [
21            println("realrobotrasp start");
22                demo consult("./mapping.pl");
23            [ !? server(ADDR,PORT) ] javaRun it.unibo.raspRobot.
                  robotClient.init(ADDR,PORT)
24        ]
25    switchTo waitForCmd
26
27        Plan waitForCmd [
28                println("realrobotrasp waiting for commands")
29        ]
30        transition stopAfter 3600000
31                whenMsg moveRobot -> execMove
32        finally repeatPlan
33
34        Plan execMove resumeLastPlan [
35        onMsg moveRobot : moveRobot(CMD) -> addRule receivedCmd(CMD);
36        [ !? mapCmd(_,OUT) ] javaRun it.unibo.raspRobot.robotClient.
              sendCmd(OUT);
```

```
37                [ ?? receivedCmd(X) ] println(removed(receivedCmd(X)))
38        ]
39 }
40
41 QActor ledagent context ctxRealRobotRasp {
42        Plan init normal [
43                println("ledagent start");
44                demo consult("./mapping.pl");
45                delay 1000 // waiting for realrobotrasp to start... (
                      robotClient.init())
46        ]
47        switchTo waitForCommand
48
49    Plan waitForCommand [ ]
50    transition stopAfter 3600000
51        whenEvent lightCmd -> switchLed
52    finally repeatPlan
53
54        Plan switchLed resumeLastPlan [
55                onEvent lightCmd : lightCmd(CMD) -> addRule receivedCmd(CMD
                      );
56                [ !? mapCmd(_,OUT) ] javaRun it.unibo.raspRobot.robotClient
                      .sendCmd(OUT);
57                [ ?? receivedCmd(X) ] println(removed(receivedCmd(X)))
58        ]
59 }
```

Listing 1.44. ../it.unibo.raspRobot/src/realrobot.qa

L'attore che rappresenta il robot utilizza un'apposita classe Java che si comporta come un client nei confronti del server Python, inviando i comandi di movimento e ricevendo da questi i dati relativi al sonar frontale.

In questo caso i comandi ricevuti vengono interpretati con l'aiuto di un file Prolog esterno, il quale contiene delle regole per effettuare il mapping dei comandi nel formato accettato dal server Python che dovrà riceverli.

L'architettura risultante è rappresentata nel dettaglio nella Fig. 12.

7.4.1 Temperature from a real sensor

Una possibile implementazione del servizio che fornisce la temperatura può essere basata su un sensore fisico collegato ad uno degli ingressi analogici di Arduino.

Anche in questo caso si tratta sempre del gestore della connessione seriale scritto in Python a fornire i dati all'esterno, mettendo a disposizione degli utilizzatori un server in grado di rispondere, su richiesta, con l'informazione della temperatura corrente.

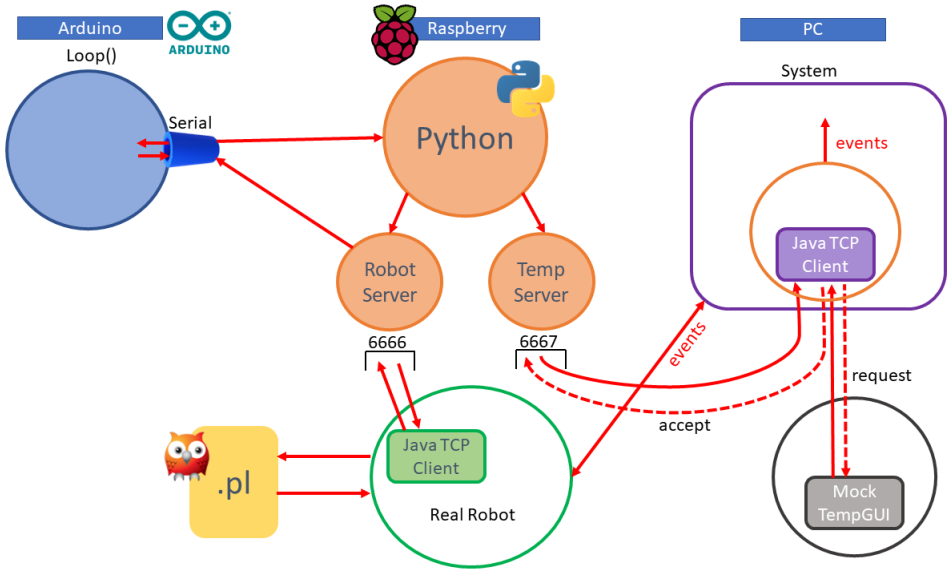


Fig. 12. Dettaglio dell'architettura del robot fisico

8 Testing

Per quanto riguarda il testing, l'approccio adottato è principalmente a *black-box*, ovvero senza assumere di avere nozione di quale sia l'effettivo comportamento interno del componente sotto esame, concentrando invece l'attenzione sull'aspetto dell'interazione. Questo consente di strutturare i test in maniera indipendente dalle specifiche implementazioni e, in alcuni casi, di riutilizzare gli stessi test durante le varie fasi dello sviluppo.

I test seguono un ordine incrementale del tipo bottom-up, partendo cioè dalle singole unità (*unit testing*), per passare poi all'integrazione di queste (*integration testing*) e, infine, ai vari sottosistemi e ai servizi da essi offerti (*functional testing*).

Queste prime tre tipologie di test possono essere eseguite in maniera automatica facendo uso del framework `JUnit`, mentre lo *user acceptance testing* viene condotto manualmente man mano che un nuovo componente viene aggiunto al sistema.

Una struttura ricorrente nei test è la verifica, in seguito al verificarsi di alcune circostanze, di cosa è presente o no nella base di conoscenza di certi attori. Nonostante per fare ciò tali QActor vengano trattati come dei semplici oggetti Java, occorre tenere presente che in realtà ognuno di essi rappresenta un flusso di controllo autonomo, pertanto è caratterizzato da tempi di avvio propri, da tempi di transizione da uno stato all'altro, eccetera.

L'utilizzo estensivo di opportune `sleep` non preclude quindi la possibilità di eventuali fallimenti dovuti all'impredicibilità dell'ordine di scheduling.

8.1 Requirements analysis testing

8.1.1 Led unit test

Il testing del led prevede di verificare semplicemente il comportamento dei metodi esposti nell'interfaccia `ILed` (Subsection 4.5). Anche se nello specifico è stato usato un oggetto mock provvisto di GUI per simulare il led fisico, il test rimane valido per qualsiasi implementazione dell'interfaccia.

```
12        private static ILed led;
13
14        @BeforeClass
15        public static void setUpBeforeClass() {
16                led = new LedMockGui();
17        }
18
19        @Test
20        public void turnOnTest() {
21                led.turnOff();
22                led.turnOn();
23                Assert.assertTrue(led.isOn());
24        }
25
26        @Test
27        public void turnOffTest() {
28                led.turnOn();
29                led.turnOff();
30                Assert.assertFalse(led.isOn());
31        }
```

Listing 1.45. ../it.unibo.finalTask2018/test/it/unibo/test/ra/LedTest.java

8.1.2 HueLampAgent unit test

L'`HueLampAgent` riceve eventi `lightCmd` e li traduce in comandi per la Hue Lamp remota. Poiché in questo caso l'oggetto del testing è il comportamento dell'agent e non la comunicazione tra questi e la Hue Lamp, possiamo limitarci a controllare che alla ricezione di un evento corrisponda un effettivo cambiamento di stato nell'oggetto Java che simula la lampada.

```
15        private static QActor huelampagentra;
16
17        @BeforeClass
18        public static void setUpBeforeClass() throws Exception {
```

```
19              MainCtxReqAnalysis.initTheContext();
20              huelampagentra = waitForQActorToStart("huelampagentra");
21      }
22
23      @Test
24      public void lightCmdReceptionTest() throws Exception {
25              huelampagentra.emit("lightCmd", "lightCmd(on)");
26              Thread.sleep(1000);
27              assertEquals("on", hueAdapter.getCurrentState());
28
29              huelampagentra.emit("lightCmd", "lightCmd(off)");
30              Thread.sleep(1000);
31              assertEquals("off", hueAdapter.getCurrentState());
32
33              huelampagentra.emit("lightCmd", "lightCmd(wrong)");
34              Thread.sleep(1000);
35              assertEquals("off", hueAdapter.getCurrentState());
36
37              huelampagentra.emit("lightCmd", "lightCmd(blink)");
38              Thread.sleep(1000);
39              assertEquals("blink", hueAdapter.getCurrentState());
40      }
```

Listing 1.46.
../it.unibo.finalTask2018/test/it/unibo/test/ra/HueLampAgentTest.java

8.1.3 TemperatureAgent unit test

Lo scopo di `TemperatureAgent` è quello di emettere periodicamente eventi `temperature`. Dobbiamo quindi verificare che, dopo un tempo ragionevole, questi siano stati effettivamente generati e che i dati sulla temperatura in essi contenuti corrispondano a quanto rilevato dal sensore.

```
15      private static QActor ralogger;
16
17      @BeforeClass
18      public static void setUpBeforeClass() throws Exception {
19              MainCtxReqAnalysis.initTheContext();
20              ralogger = waitForQActorToStart("ralogger");
21      }
22
23      @Test
24      public void temperatureEmissionTest() throws Exception {
25              Thread.sleep(5000 + 1000);
26              assertTrue(isEventReceived(ralogger, "temperature", "
                    temperature(" + temperatureAdapter.getCurrentTemp() +
                    ")"));
27      }
```

Listing 1.47. ../it.unibo.finalTask2018/test/it/unibo/test/ra/TemperatureAgentTest.java

Per registrare gli eventi emessi nel sistema è stato usato un *event logger* event-driven:

```
12 EventHandler evhralogger for clock, temperature, sonarSensor, frontSonar,
       robotCmd, lightCmd /*-print*/ {
13         memoCurrentEvent -lastonly for ralogger;
14         demo dologevent for ralogger
15 };
```

Listing 1.48. ../it.unibo.finalTask2018/src/requirementsAnalysis.qa

```
123 QActor ralogger context ctxReqAnalysis {
124         Plan init normal [
125                 demo consult("./logger.pl")
126         ]
127         transition stopAfter 3600000
128         finally repeatPlan
129 }
```

Listing 1.49. ../it.unibo.finalTask2018/src/requirementsAnalysis.qa

Il comportamento da attuare alla ricezione di un evento è precisato in un file Prolog esterno, così che possa essere condiviso da più event logger.

```
1 dologevent :-
2         msg(EVID,'event',EMITTER,none,EVPL,NUM),
3         assert(logevent(EVID,EVPL)).
4         % actorPrintln(loggedev(EVID,EVPL)).
```

Listing 1.50. ../it.unibo.finalTask2018/logger.pl

8.1.4 ClockAgent unit test

Un ragionamento analogo al precedente vale anche per il testing di `ClockAgent`:

```
15         private static QActor ralogger;
16 
17         @BeforeClass
18         public static void setUpBeforeClass() throws Exception {
19                 MainCtxReqAnalysis.initTheContext();
20                 ralogger = waitForQActorToStart("ralogger");
21         }
```

```
22
23      @Test
24      public void clockEmissionTest() throws Exception {
25              Thread.sleep(6000 + 1000);
26              int[] time = clockAdapter.getCurrentTime();
27              assertTrue(isEventReceived(ralogger, "clock", "clock(" +
                    time[0] + "," + time[1] + ")"));
28      }
```

Listing 1.51.
../it.unibo.finalTask2018/test/it/unibo/test/ra/ClockAgentTest.java

8.1.5 HumanOperator and Application logic integration test

Poiché lo `HumanOperator` è un attore che emette comandi per il robot nella forma di messaggi per il componente che incapsula la logica applicativa, il testing del primo è legato a quello della seconda.

Nello specifico, occorre verificare che quest'ultima emetta i rispettivi eventi `robotCmd` per il robot e `lightCmd` per il led/la Hue Lamp. A tal scopo ricorriamo ancora una volta all'event logger.

```
14      private static QActor ralogger;
15      private static QActor humanoperatorra;
16
17      @BeforeClass
18      public static void setUpBeforeClass() throws Exception {
19              MainCtxReqAnalysis.initTheContext();
20              ralogger = waitForQActorToStart("ralogger");
21              humanoperatorra = waitForQActorToStart("humanoperatorra");
22      }
23
24      @Test
25      public void robotCmdEmissionTest() throws Exception {
26              Thread.sleep(2000);
27              assertTrue(isEventReceived(ralogger, "robotCmd", "moveRobot
                    (w(X))"));
28              assertTrue(isEventReceived(ralogger, "robotCmd", "moveRobot
                    (h(X))"));
29      }
30
31      @Test
32      public void lightCmdEmissionTest() throws Exception {
33              Thread.sleep(5000);
34              assertTrue(isEventReceived(ralogger, "lightCmd", "lightCmd(
                    on)"));
35              assertTrue(isEventReceived(ralogger, "lightCmd", "lightCmd(
                    off)"));
36      }
```

```
37
38      @Test
39      public void cmdMsgEmissionTest() throws Exception {
40              sendMsg(humanoperatorra, "applra", "cmd", "cmd(d(0))");
41              Thread.sleep(2000);
42              assertTrue(isEventReceived(ralogger, "robotCmd", "moveRobot
                    (d(X))"));
43      }
```

Listing 1.52.
../it.unibo.finalTask2018/test/it/unibo/test/ra/HumanOpApplTest.java

Si noti come i primi due test facciano riferimento ai messaggi inviati a fine di testing direttamente nel modello dell'analisi dei requisiti (Subsection 4.7).

8.1.6 Robot unit test

Il robot, sia fisico sia virtuale, è stato modellato come un attore in grado di interpretare messaggi `moveRobot` e tradurli in opportune azioni di movimento. Oltre a questo, il robot è sensibile agli eventi `robotCmd` generati dalla logica applicativa e mappati in messaggi da un `EventHandler`.

Infine, il robot può emettere informazioni relative al sonar frontale e al proprio passaggio in corrispondenza di uno dei due sonar presenti nella stanza.

Il testing si concentrerà quindi su questi tre aspetti, verificando lo stato del robot in seguito all'invio di comandi sotto forma di messaggi ed eventi, e consultando la base di conoscenza dell'event logger per controllare l'effettiva emissione degli eventi `frontSonar` e `sonarSensor`.

```
17      private static QActor ddrlogger;
18
19      @BeforeClass
20      public static void setUpBeforeClass() throws Exception {
21              MainCtxDdr.initTheContext();
22              ddrlogger = waitForQActorToStart("ddrlogger");
23      }
24
25      @Test
26      public void moveRobotReceptionTest() throws Exception {
27              sendMsg(ddrlogger, "ddr", "moveRobot", "moveRobot(d(0))");
28              Thread.sleep(2000);
29              assertEquals("right", robotAdapter.getStatus());
30      }
31
32      @Test
33      public void robotCmdReceptionTest() throws Exception {
34              ddrlogger.emit("robotCmd", "moveRobot(a(0))");
35              Thread.sleep(2000);
```

```
36              assertTrue(isEventReceived(ddrlogger, "robotCmd", "
                    moveRobot(a(0))"));
37              assertEquals("left", robotAdapter.getStatus());
38      }
39
40      @Test
41      public void sonarEmissionTest() throws Exception {
42              Thread.sleep(5000);
43              assertTrue(isEventReceived(ddrlogger, "sonarSensor", "sonar
                    (N,D)"));
44              assertTrue(isEventReceived(ddrlogger, "frontSonar", "sonar(
                    D)"));
45      }
```

Listing 1.53. ../it.unibo.finalTask2018/test/it/unibo/test/ra/DdrTest.java

8.1.7 Robot & Application system integration test

Dopo aver testato che l'applicazione sia in grado di tradurre messaggi `cmd` in opportuni eventi `robotCmd` e `lightCmd`, e che il robot interpreti correttamente tali comandi, rimane da verificare che i due componenti continuino a funzionare una volta entrati a far parte di uno stesso sistema (Subsection 4.10).

Per automatizzare il test occorre eseguire in processi distinti l'applicazione principale, il robot e, infine, il "system integrator" tra gli altri due.

In particolare, l'obiettivo è controllare che all'invio di un `cmd` diretto al componente che incapsula la logica applicativa, il robot mock agisca di conseguenza grazie alla propagazione degli eventi garantita dall'infrastruttura *QActor*.

```
20      private static QActor ddrlogger;
21
22      private static Process integrator;
23      private static Process reqanal;
24      private static QActorContext ctxDdr;
25
26      @BeforeClass
27      public static void setUpBeforeClass() throws Exception {
28              ctxDdr = MainCtxDdr.initTheContext();
29              ddrlogger = waitForQActorToStart("ddrlogger");
30
31              reqanal = execMain(MainCtxReqAnalysis.class);
32              System.out.println("Waiting for MainCtxDdr and
                    MainCtxReqAnalysis to start...");
33              Thread.sleep(20000);
34              integrator = execMain(MainCtxRaIntegrator.class);
35              Thread.sleep(10000);
36      }
```

```
37
38      @AfterClass
39      public static void tearDownAfterClass() throws Exception {
40              System.out.println("Terminating QActor system MainCtxDdr");
41              Thread.sleep(5000);
42              ctxDdr.terminateQActorSystem();
43              Thread.sleep(5000);
44
45              stopProcess(integrator);
46              System.out.println("MainCtxRaIntegrator stopped");
47
48              stopProcess(reqanal);
49              System.out.println("MainCtxReqAnalysis stopped");
50      }
51
52      @Test
53      public void cmdReceptionTest() throws Exception {
54              sendMsg(ddrlogger, "applra", "cmd", "cmd(w(2))");
55              Thread.sleep(2000);
56              assertTrue(isEventReceived(ddrlogger, "robotCmd", "
                    moveRobot(w(2))"));
57              assertEquals("forward", robotAdapter.getStatus());
58      }
59
60      @Test
61      public void lightCmdEmissionTest() throws Exception {
62              sendMsg(ddrlogger, "applra", "cmd", "cmd(h(2))");
63              Thread.sleep(2000);
64              assertTrue(isEventReceived(ddrlogger, "lightCmd", "lightCmd
                    (off)"));
65      }
```

Listing 1.54. ../it.unibo.finalTask2018/test/it/unibo/test/ra/DdrApplIntegrationTest.java

8.2 Problem analysis testing

8.2.1 Controller unit test

Gran parte della logica del `Controller` è espressa sotto forma di regole Prolog inserite all'interno della base di conoscenza del relativo attore. Per testare queste regole dobbiamo quindi, come fatto in precedenza, sfruttare i metodi dell'oggetto Java corrispondente.

```
15      private static QActor controller;
16      private static QActor logger;
17
18      @BeforeClass
```

```
19          public static void setUpBeforeClass() throws Exception {
20                  MainCtxApplPA.initTheContext();
21                  controller = waitForQActorToStart("controllerpa");
22                  logger = waitForQActorToStart("palogger");
23          }
24
25          @Test
26          public void changeGetModelItemTest() throws Exception {
27                  String t = "10";
28                  contrChangeModel("temperature,t1," + t);
29                  assertTrue(contrGetModel("t1," + t).isSuccess());
30
31                  String h = "13";
32                  String m = "30";
33                  contrChangeModel("clock,c1,h(" + h + "," + m + ")");
34                  assertTrue(contrGetModel("c1,h(" + h + "," + m + ")").
                        isSuccess());
35          }
36
37          @Test
38          public void overLimitTempTest() throws Exception {
39                  contrChangeModel("robot,r1,w(0)");
40                  assertTrue(contrGetModel("r1,w(0)").isSuccess());
41
42                  String t = contrSolve("limitTemperatureValue(T)").getTerm("
                        T").toString();
43                  contrChangeModel("temperature,t1," + t);
44                  assertTrue(contrGetModel("r1,h(X)").isSuccess());
45          }
46
47          @Test
48          public void outOfBoundaryTimeTest() throws Exception {
49                  contrChangeModel("robot,r1,w(0)");
50                  assertTrue(contrGetModel("r1,w(0)").isSuccess());
51
52                  SolveInfo si = contrSolve("timeInterval(h(_,_),h(H,M))");
53                  String h = si.getTerm("H").toString();
54                  String m = si.getTerm("M").toString();
55                  contrChangeModel("clock,c1,h(" + h + "," + m + ")");
56                  assertTrue(contrGetModel("r1,h(X)").isSuccess());
57          }
58
59          @Test
60          public void changeLedStateTest() throws Exception {
61                  contrChangeModel("robot,r1,h(0)");
62                  contrChangeModel("robot,r1,w(0)");
63                  assertTrue(contrGetModel("l1,blink").isSuccess());
64
65                  contrChangeModel("robot,r1,h(0)");
66                  assertTrue(contrGetModel("l1,off").isSuccess());
```

```
67          }
68
69          @Test
70          public void leftRightTurnTest() throws Exception {
71                  contrChangeModel("robot,r1,a(0)");
72                  assertTrue(contrGetModel("r1,h(X)").isSuccess());
73
74                  contrChangeModel("robot,r1,d(0)");
75                  assertTrue(contrGetModel("r1,h(X)").isSuccess());
76          }
77
78          @Test
79          public void robotCanMoveTest() throws Exception {
80                  boolean canMove = contrSolve("canMove").isSuccess();
81                  if(!canMove)
82                          controller.addRule("canMove");
83                  contrChangeModel("robot,r1,h(0)");
84                  contrSolve("changeRobotModel(w(0))");
85                  assertTrue(contrGetModel("r1,w(X)").isSuccess());
86                  if(!canMove)
87                          controller.removeRule("canMove");
88
89                  if(contrSolve("canMove").isSuccess()) {
90                          // temperatura non valida
91                          String t = contrSolve("limitTemperatureValue(T)").
                                getTerm("T").toString();
92                          contrChangeModel("temperature,t1," + t);
93                  }
94                  contrChangeModel("robot,r1,h(0)");
95                  contrSolve("changeRobotModel(w(0))");
96                  assertTrue(contrGetModel("r1,h(X)").isSuccess());
97          }
```

Listing 1.55.
../it.unibo.finalTask2018/test/it/unibo/test/pa/ControllerTest.java

Un'altra serie di test riguardano l'aspetto dell'interazione. Nell'effettivo, il `Controller` interagisce con l'esterno tramite gli eventi `ctrlEvent` (alcuni dei quali sono convertiti in `lightCmd` da un apposito EventHandler), `inputCtrlEvent` e i messaggi `cmd`.

```
99          @Test
100         public void ctrlEventRobotEmissionTest() throws Exception {
101                 contrChangeModel("robot,r1,w(0)");
102                 contrChangeModel("robot,r1,h(0)");
103                 Thread.sleep(1000);
104                 assertTrue(isEventReceived(logger, "ctrlEvent", "ctrlEvent(
                        robot,r1,h(0))"));
105         }
```

```
106
107         @Test
108         public void lightCmdEmissionTest() throws Exception {
109                 contrChangeModel("led,l1,on");
110                 contrChangeModel("led,l1,off");
111                 Thread.sleep(1000);
112                 assertTrue(isEventReceived(logger, "lightCmd", "lightCmd(
                        off)"));
113         }
114
115         @Test
116         public void inputCtrlEventReceptionTest() throws Exception {
117                 contrChangeModel("led,l1,off");
118                 controller.emit("inputCtrlEvent", "inputEvent(led,l1,on)");
119                 Thread.sleep(1000);
120                 assertTrue(contrGetModel("l1,on").isSuccess());
121         }
122
123         @Test
124         public void cmdMsgReceptionTest() throws Exception {
125                 contrChangeModel("robot,r1,h(0)");
126                 controller.addRule("canMove");
127
128                 sendMsg(logger, controller, "cmd", "cmd(w(0))");
129                 Thread.sleep(1000);
130
131                 assertTrue(contrGetModel("r1,w(0)").isSuccess());
132                 controller.removeRule("canMove");
133         }
```

Listing 1.56.
../it.unibo.finalTask2018/test/it/unibo/test/pa/ControllerTest.java

8.2.2 Software Agent unit test

Per il testing di `SoftwareAgent` occorre verificare che, in seguito all'emissione degli eventi generati dai sonar (quello frontale e i due fissi), vengano recapitati al controller opportuni messaggi `cmd`. L'effetto di questi si ripercuote poi sullo stato del modello del robot e sugli eventi `ctrlEvent` emessi per pilotarlo.

```
14         private static QActor controller;
15         private static QActor swagent;
16         private static QActor logger;
17
18         @BeforeClass
19         public static void setUpBeforeClass() throws Exception {
20                 MainCtxApplPA.initTheContext();
21                 controller = waitForQActorToStart("controllerpa");
```

```
22              swagent = waitForQActorToStart("swagpa");
23              logger = waitForQActorToStart("palogger");
24
25              // ok modifiche al modello del robot
26              controller.addRule("canMove");
27              swagent.addRule("isClose");
28      }
29
30      @Test
31      public void frontSonarCmdEmissionTest() throws Exception {
32              Thread.sleep(1000);
33              controller.solveGoal("changeModelItem(robot,r1,w(0))");
34              controller.emit("frontSonar", "sonar(2)");
35              Thread.sleep(400);
36              assertTrue(controller.solveGoal("getModelItem(r1,h(X))").
                    isSuccess());
37      }
38
39      @Test
40      public void sonarSensorCmdEmissionTest() throws Exception {
41              Thread.sleep(1000);
42              controller.solveGoal("changeModelItem(robot,r1,h(0))");
43              controller.emit("sonarSensor", "sonar(sonar1,2)");
44              Thread.sleep(400);
45              assertTrue(isEventReceived(logger, "ctrlEvent", "ctrlEvent(
                    robot,r1,d(0))"));
46      }
```

Listing 1.57.
../it.unibo.finalTask2018/test/it/unibo/test/pa/SwAgentTest.java

8.2.3 HumanOperator unit test

Un ragionamento simile può essere fatto per `HumanOperator`: questa volta è all'emissione di eventi `usercmd` – gli stessi originati dalla web GUI – che deve seguire l'invio dei relativi messaggi `cmd`.

```
14      private static QActor controller;
15
16      @BeforeClass
17      public static void setUpBeforeClass() throws Exception {
18              MainCtxApplPA.initTheContext();
19              controller = waitForQActorToStart("controllerpa");
20              controller.addRule("canMove");
21      }
22
23      @Test
24      public void cmdEmissionTest() throws Exception {
```

```
25              Thread.sleep(1000);
26              controller.emit("usercmd", "usercmd(robotgui(w(0)))");
27              Thread.sleep(400);
28              assertTrue(controller.solveGoal("getModelItem(r1,w(0))").
                    isSuccess());
29
30              controller.emit("usercmd", "usercmd(robotgui(h(0)))");
31              Thread.sleep(400);
32              assertTrue(controller.solveGoal("getModelItem(r1,h(0))").
                    isSuccess());
33      }
```

Listing 1.58.
../it.unibo.finalTask2018/test/it/unibo/test/pa/HumanOpTest.java

8.2.4 Architecture integration test

Dopo aver testato separatamente i vari componenti definiti nell'analisi del problema, passiamo a testare il sistema nel suo complesso. In particolare, siamo interessati a verificare che l'introduzione del resource model e del relativo controller abbia rispettato i quattro punti descritti nella Section 5.

In altre parole, simulando il cambiamento di stato di un sensore ci aspettiamo che questo venga correttamente notificato al controller, così che questi possa aggiornarne il modello. Poi, in accordo con la logica applicativa, deve seguire una modifica del modello degli attuatori, la loro notifica da parte del controller e un effettivo riscontro sugli attuatori "reali".

```
20      private static QActor controller;
21      private static Process robot;
22
23      @BeforeClass
24      public static void setUpBeforeClass() throws Exception {
25              MainCtxApplPA.initTheContext();
26              controller = waitForQActorToStart("controllerpa");
27
28              robot = execMain(MainCtxRealRobotMock.class);
29              Thread.sleep(10000);
30      }
31
32      @AfterClass
33      public static void tearDownAfterClass() throws Exception {
34              stopProcess(robot);
35              System.out.println("MainCtxRealRobotMock stopped");
36      }
37
38      @Test
39      public void sensorsControllerActuatorsTest() throws Exception {
```

```
40              TCPClient rClient = new TCPClient(8999);
41              BufferedReader robotReader = rClient.getBufferedReader();
42              Thread.sleep(5000);
43
44              // usercmd -> robot: forward, led: blink
45              controller.addRule("canMove");
46              controller.emit("usercmd", "usercmd(robotgui(w(4)))");
47              Thread.sleep(2000);
48
49              assertEquals("blink", lampAdapter.getState());
50
51              rClient.writeLine("getState");
52              assertEquals("forward", robotReader.readLine());
53              rClient.close();
54      }
```

Listing 1.59.
../it.unibo.finalTask2018/test/it/unibo/test/pa/ArchIntegrationTest.java

In questo caso si è simulata la pressione di un pulsante sulla console, per poi controllare sia lo stato della lampada, sia quello del robot, in entrambi i casi facendo uso di oggetti mock. Da notare come quest'ultimo, poiché si trova in un contesto diverso, offra, a fine di testing, un semplice server TCP per permettere di verificare dall'esterno il proprio stato.

8.2.5 Time, temperature and blinking functional test

Il passo successivo è testare la correttezza dei requisiti ad ora soddisfatti: R-TempOk, R-TempKo, R-TimeOk, R-TimeKo, R-BlinkLed e R-BlinkHue.

Per farlo inviamo una serie di comandi di movimento al robot e, sotto opportune condizioni, verifichiamo che:

- Il robot si muova quando la temperatura è al di sotto della soglia (e l'orario entro l'intervallo), viceversa non si deve muovere (R-TempOk).
- Il robot si muova quando l'orario è all'interno dell'intervallo (e la temperatura al di sotto della soglia), viceversa non si deve muovere (R-TimeOk).
- Non appena la temperatura sale al di sopra della soglia, il robot deve fermarsi (R-TempKo).
- Non appena l'orario esce dall'intervallo consentito, il robot deve fermarsi (R-TimeKo).
- La lampada deve lampeggiare quando il robot è in movimento, altrimenti deve essere spenta (R-BlinkHue).

```
22      private static QActor controller;
23      private static Process robot;
24
25      private static TCPClient rClient;
26      private static BufferedReader robotReader;
```

```
27
28      @BeforeClass
29      public static void setUpBeforeClass() throws Exception {
30              MainCtxApplPA.initTheContext();
31              controller = waitForQActorToStart("controllerpa");
32
33              robot = execMain(MainCtxRealRobotMock.class);
34              Thread.sleep(10000);
35              rClient = new TCPClient(8999);
36              robotReader = rClient.getBufferedReader();
37      }
38
39      @AfterClass
40      public static void tearDownAfterClass() throws Exception {
41              rClient.close();
42              stopProcess(robot);
43              System.out.println("MainCtxRealRobotMock stopped");
44      }
45
46      @Before
47      public void beforeTest() {
48              // time e temperature ok
49              setTempLimit(100);
50              setTimeInterval(0, 100);
51      }
52
53      @Test
54      public void temperatureOkKoTest() throws Exception {
55              controller.solveGoal("changeModelItem(robot,r1,h(0))");
56
57              // TempOk: t<limit, can move
58              controller.emit("usercmd", "usercmd(robotgui(w(4)))");
59              Thread.sleep(2000);
60              assertEquals("blink", lampAdapter.getState());
61              assertEquals("forward", getRobotState());
62
63              // TempKo: stop
64              setTempLimit(0);
65              controller.solveGoal("getModelItem(t1,T), T2 is T+1,"
66                              + "changeModelItem(temperature,t1,T2)");
67              Thread.sleep(2000);
68              assertEquals("off", lampAdapter.getState());
69              assertEquals("stop", getRobotState());
70
71              // TempOk: t>=limit, cannot move
72              controller.emit("usercmd", "usercmd(robotgui(w(4)))");
73              Thread.sleep(2000);
74              assertEquals("off", lampAdapter.getState());
75              assertEquals("stop", getRobotState());
76      }
```

```
77
78      @Test
79      public void timeOkKoTest() throws Exception {
80              controller.solveGoal("changeModelItem(robot,r1,h(0))");
81
82              // TimeOk: lbound<t<hbound, can move
83              controller.emit("usercmd", "usercmd(robotgui(w(4)))");
84              Thread.sleep(2000);
85              assertEquals("blink", lampAdapter.getState());
86              assertEquals("forward", getRobotState());
87
88              // TimeKo: stop
89              setTimeInterval(0, 0);
90              controller.solveGoal("getModelItem(c1,h(H,_)), H2 is H+1,"
91                              + "changeModelItem(clock,c1,h(H2,0))");
92              Thread.sleep(2000);
93              assertEquals("off", lampAdapter.getState());
94              assertEquals("stop", getRobotState());
95
96              // TimeOk: t>=hbound, cannot move
97              controller.emit("usercmd", "usercmd(robotgui(w(4)))");
98              Thread.sleep(2000);
99              assertEquals("off", lampAdapter.getState());
100             assertEquals("stop", getRobotState());
101     }
```

Listing 1.60. ../it.unibo.finalTask2018/test/it/unibo/test/pa/TimeTempFunctionalTest.java

8.3 Project testing

La parte di progetto si concentra sulla realizzazione dei requisiti relativi alla pulizia della stanza ad opera di `SoftwareAgent`, affrontandoli progressivamente in maniera incrementale. Anche i test seguiranno quindi il medesimo approccio per gradi, impiegando versioni dell'agent con i soli stati necessari.

8.3.1 Start/Stop/End test

Il testing dei requisiti legati all'avvio e alla terminazione della pulizia sono effettuati emettendo gli stessi eventi generati dalla console e dai sonar fissi, verificando che solo in presenza delle giuste condizioni l'agent transiti negli stati corretti. Ciò è possibile grazie al mantenimento dello stato corrente nella base di conoscenza dell'attore.

```
18      private static QActor swag;
19
20      @BeforeClass
```

```
21          public static void setUpBeforeClass() throws Exception {
22                  MainCtxSwag1.initTheContext();
23                  swag = waitForQActorToStart("swag1");
24          }
25
26          @After
27          public void afterTest() {
28                  swag.solveGoal("removeLogMsg");
29          }
30
31          @Test
32          public void externalCmdReceptionTest() throws Exception {
33                  swag.emit("alarm", "usercmd(testcmd)");
34                  Thread.sleep(1000);
35                  assertTrue(isMsgReceived(swag, "externalcmd", "usercmd(
                        testcmd)"));
36          }
37
38          @Test
39          public void cleanCmdTooLateSonarTest() throws Exception {
40                  swag.emit("alarm", "usercmd(clean)");
41                  Thread.sleep(2000);
42                  swag.emit("sonarSensor", "sonar(sonar1,1)");
43                  // too late
44                  assertEquals("init", getCurrentState());
45
46                  swag.emit("alarm", "usercmd(clean)");
47                  Thread.sleep(1000); // sonarSensor lost?
48                  swag.emit("alarm", "usercmd(halt)");
49                  Thread.sleep(1000);
50          }
51
52          @Test
53          public void cleanCmdNotCloseToSonarTest() throws Exception {
54                  swag.emit("alarm", "usercmd(clean)");
55                  Thread.sleep(400);
56                  swag.emit("sonarSensor", "sonar(sonar1,10)");
57                  Thread.sleep(2000);
58                  // not close to sonar1
59                  assertEquals("init", getCurrentState());
60          }
61
62          @Test
63          public void cleanHaltCmdTest() throws Exception {
64                  swag.emit("alarm", "usercmd(clean)");
65                  Thread.sleep(400);
66                  swag.emit("sonarSensor", "sonar(sonar1,1)");
67                  Thread.sleep(400);
68                  // ok clean
69                  assertEquals("cleaning", getCurrentState());
```

```
70                assertTrue(isMsgReceived(swag, "swagmsg", "cmd(clean)"));
71
72                swag.emit("alarm", "usercmd(halt)");
73                Thread.sleep(2000);
74                // halted
75                assertEquals("init", getCurrentState());
76                assertTrue(isMsgReceived(swag, "externalcmd", "usercmd(halt
                      )"));
77        }
78
79        @Test
80        public void endTest() throws Exception {
81                swag.emit("alarm", "usercmd(clean)");
82                Thread.sleep(400);
83                swag.emit("sonarSensor", "sonar(sonar1,1)");
84                Thread.sleep(400);
85                // ok
86                assertEquals("cleaning", getCurrentState());
87                assertTrue(isMsgReceived(swag, "swagmsg", "cmd(clean)"));
88
89                swag.emit("sonarSensor", "sonar(sonar2,10)");
90                Thread.sleep(2000);
91                // not close to sonar2
92                assertEquals("cleaning", getCurrentState());
93
94                swag.emit("sonarSensor", "sonar(sonar2,1)");
95                Thread.sleep(400);
96                // ok end
97                assertEquals("init", getCurrentState());
98                assertTrue(isMsgReceived(swag, "swagmsg", "cmd(halt)"));
99        }
```

Listing 1.61. ../it.unibo.finalTask2018/test/it/unibo/test/crs/SwAgentStartStopEndTest.java

```
1  setCurrentState(S) :-
2          replaceRule( currentState(_), currentState(S) ).
3
4  removeLogMsg :-
5          logmsg(NAME,PL),
6          retract(logmsg(NAME,PL)),
7          removeLogMsg.
```

Listing 1.62. swag1 Rules

8.3.2 Cleaning the floor test

Per testare gli stati che corrispondono alla pulizia a "zig-zag", la versione di `SoftwareAgent` sotto esame riceve direttamente dal codice del test gli eventi relativi ai vari sonar, così da non dover mettere in campo alcun ambiente virtuale.

Come stabilito in fase di analisi, poiché abbiamo adottato questa strategia di pulizia, quando il robot viene rilevato vicino a sonar2 possiamo assumere di aver pulito l'intera superficie.

```
14      private static QActor swag;
15
16      @BeforeClass
17      public static void setUpBeforeClass() throws Exception {
18              MainCtxSwag3.initTheContext();
19              swag = waitForQActorToStart("swag3");
20      }
21
22      @Test
23      public void floorCleaningTest() throws Exception {
24              // startCleaning
25              simulateWall();
26              // leftTurn
27              Thread.sleep(5000);
28              // backCleaning
29              simulateWall();
30              // rightTurn
31              Thread.sleep(5000);
32              // forwardCleaning
33              simulateSonar2(20);
34
35              // leftTurn
36              Thread.sleep(5000);
37              // backCleaning
38              simulateWall();
39              // rightTurn
40              Thread.sleep(5000);
41              // forwardCleaning
42              simulateSonar2(2);
43              // end
44              Thread.sleep(400);
45
46              assertTrue(swag.solveGoal("stateHistory(["
47                      + "startCleaning,leftTurn,backCleaning,rightTurn,
                            forwardCleaning,"
48                      + "leftTurn,backCleaning,rightTurn,forwardCleaning,
                            end])").isSuccess());
49      }
```

Listing 1.63.
../it.unibo.finalTask2018/test/it/unibo/test/crs/SwAgentCleaningTest.java

Questa versione dell'agent, a fine di testing, tiene traccia nella propria base di conoscenza di tutti gli stati in cui è transitato, così da poterne verificare la

corretta sequenza.

```
addState(S) :-
        findall(X,logstate(X,Y),LIST),
        length(LIST,N),
        assert(logstate(S,N)).

stateHistory(LIST) :- stateHistory(LIST,0).

stateHistory([],N) :- not(logstate(_,N)).

stateHistory([H|T],N) :-
        logstate(H,N),
        N2 is N+1,
        stateHistory(T,N2).

removeLogState :-
        logstate(S,N),
        retract(logstate(S,N)),
        removeLogState.
```

Listing 1.64. ../it.unibo.finalTask2018/swagtest.pl

8.3.3 Obstacle avoidance test

L'obiettivo di questi test è verificare i requisiti che riguardano il superamento degli ostacoli fissi e mobili e l'eventuale rinuncia in caso di un ostacolo "invalicabile". In particolare, l'agent:

- Quando riceve un solo `frontSonar`, deve identificarlo come un ostacolo mobile
- Quando ne riceve due a breve distanza, deve riconoscere l'ostacolo come fisso ed aggirarlo da destra
- Se mentre prova a passarlo da destra non trova subito un varco, deve ritentare più in là
- Se durante la ricerca di un passaggio incontra un ostacolo, deve tornare indietro e tentare l'aggiramento da sinistra
- Se finisce i tentativi di esplorazione a destra, prova a sinistra; se fallisce nuovamente, rinuncia

```
        private static QActor swag;

        @BeforeClass
        public static void setUpBeforeClass() throws Exception {
                MainCtxSwag4.initTheContext();
                swag = waitForQActorToStart("swag4");
                Thread.sleep(2000);
        }
```

```
24
25          @After
26          public void afterTest() throws Exception {
27                  String currState = getLastState();
28                  swag.solveGoal("removeLogState");
29
30                  if(currState != null) {
31  //  System.out.println("Current state = " + currState);
32                          swag.addRule("logstate(" + currState + ",0)");
33                  }
34          }
35
36          @Test
37          public void avoidMobileTest() throws Exception {
38                  // cleaning
39                  simulateObstacle();
40                  // handleFront
41                  Thread.sleep(2000);
42                  // avoidMobile
43                  // cleaning
44
45                  assertHistory("[cleaning,handleFront,avoidMobile,cleaning]"
                        );
46          }
47
48          @Test
49          public void avoidFixDoorFoundRightTest() throws Exception {
50                  // cleaning
51                  simulateObstacle();
52                  // handleFront
53                  Thread.sleep(400);
54                  simulateObstacle();
55                  // avoidFix
56                  Thread.sleep(20000);
57                  // checkDoor
58                  // doorFound
59                  // goToPrevLevel
60                  // goToPrevLevel (riprendo direzione di marcia)
61                  // cleaning
62
63                  assertHistory("[cleaning,handleFront,avoidFix,"
64                                  + "checkDoor,doorFound,goToPrevLevel,
                                        goToPrevLevel,cleaning]");
65          }
66
67          @Test
68          public void avoidFixDoorFoundRight2Test() throws Exception {
69                  // cleaning
70                  simulateObstacle();
71                  // handleFront
```

```
72              Thread.sleep(1200);
73              simulateObstacle();
74              // avoidFix
75              Thread.sleep(400);
76              assertEquals("avoidFix", getLastState());
77              Thread.sleep(2000);
78
79              // il frontSonar deve arrivare in checkDoor
80              // se arriva in avoidFix va in failure
81              waitForState("checkDoor");
82
83              // checkDoor
84              simulateObstacle();
85              // avoidFix
86              // checkDoor
87              // doorFound
88              // goToPrevLevel
89              // goToPrevLevel
90              // goToPrevLevel (riprendo direzione di marcia)
91              // cleaning
92              Thread.sleep(25000);
93
94              assertHistory("[cleaning,handleFront,avoidFix,checkDoor,
                    avoidFix,"
95                      + "checkDoor,doorFound,goToPrevLevel,goToPrevLevel,
                            goToPrevLevel,cleaning]");
96      }
97
98      @Test
99      public void avoidFixFailureRightTest() throws Exception {
100             // cleaning
101             simulateObstacle();
102             // handleFront
103             Thread.sleep(1200);
104             simulateObstacle();
105             Thread.sleep(400);
106             // avoidFix
107             assertEquals("avoidFix", getLastState());
108             Thread.sleep(5400);
109             simulateObstacle();
110             Thread.sleep(400);
111             // failure
112             assertEquals("failure", getLastState());
113             // giveUp
114             // resumeLastPosition
115             // avoidFix
116             // checkDoor
117             // doorFound
118             // goToPrevLevel
119             // goToPrevLevel (riprendo direzione di marcia)
```

```
120                 // cleaning
121                 Thread.sleep(25000);
122 
123                 assertHistory("[cleaning,handleFront,avoidFix,failure,
                        giveUp,"
124                         + "resumeLastPosition,avoidFix,checkDoor,doorFound,
                            goToPrevLevel,goToPrevLevel,cleaning]");
125         }
126 
127         @Test
128         public void avoidFixGiveUpRightLeftTest() throws Exception {
129                 String limit = swag.solveGoal("giveUpLimit(L)").getTerm("L"
                        ).toString();
130                 swag.replaceRule("giveUpLimit(_)","giveUpLimit(1)");
131 
132                 // cleaning
133                 simulateObstacle();
134                 // handleFront
135                 Thread.sleep(1200);
136                 simulateObstacle();
137                 // avoidFix
138                 Thread.sleep(800);
139                 // giveUp
140                 // resumeLastPosition
141                 // avoidFix
142                 // giveUp
143                 // (init)
144                 // cleaning
145                 Thread.sleep(20000);
146 
147                 swag.replaceRule("giveUpLimit(_)","giveUpLimit(" + limit +
                        ")");
148 
149                 assertHistory("[cleaning,handleFront,avoidFix,giveUp,"
150                         + "resumeLastPosition,avoidFix,giveUp,cleaning]");
151         }
```

Listing 1.65.
../it.unibo.finalTask2018/test/it/unibo/test/crs/SwAgentObstacleAvTest.java

8.4 Implementation testing

Poiché i modelli dei vari componenti sono già stati testati nelle sezioni precedenti, per verificarne l'implementazione relativa ad una certa tecnologia ci basta strutturare dei test che coprano i metodi Java esposti ed impiegati tramite `javaRun`.

8.4.1 Hue Lamp Agent interaction test

L'implementazione dello `HueLampAgent` fa uso di una classe Java per comunicare in modo RESTful con il bridge e la *smart lamp*. In assenza di una Hue Lamp fisica, viene utilizzato un server HTTP che la simuli.

```
21      private static Process lamp;
22
23      @BeforeClass
24      public static void setUpBeforeClass() throws Exception {
25              lamp = Runtime.getRuntime().exec("java -jar ../
                    HueLampMockServer.jar");
26              Thread.sleep(10000);
27      }
28
29      @AfterClass
30      public static void tearDownAfterClass() throws Exception {
31              if(lamp!=null && lamp.isAlive())
32                      lamp.destroyForcibly();
33      }
34
35      @Test
36      public void setUpTest() throws Exception {
37              File settings = new File(hueLampAdapter.FILE_NAME);
38              if(settings.exists())
39                      settings.delete();
40
41              hueLampAdapter.setUp(null);
42
43              try(BufferedReader br = new BufferedReader(new FileReader(
                    hueLampAdapter.FILE_NAME))) {
44                      String bridge = br.readLine();
45                      String username = br.readLine();
46                      String lampID = br.readLine();
47
48                      assertEquals(hueLampAdapter.DEFAULT_BRIDGE, bridge)
                            ;
49                      assertEquals("fnp", username);
50                      assertEquals("1", lampID);
51              }
52      }
53
54      @Test
55      public void turnOnTest() throws Exception {
56              hueLampAdapter.setUp(null);
57              hueLampAdapter.setLampState(null, "on");
58              HttpResponse response = RESTfulClient.execGET("http://"+
                    hueLampAdapter.DEFAULT_BRIDGE+"/api/fnp/lights/1/state
                    /on");
```

```
59              JSONObject obj = JSONParser.parseJSONObject(response.
                    getEntity().getContent());
60              assertEquals("{\"effect\":\"none\",\"on\":true}", obj.
                    toString());
61      }
62
63      @Test
64      public void turnOffTest() throws Exception {
65              hueLampAdapter.setUp(null);
66              hueLampAdapter.setLampState(null, "off");
67              HttpResponse response = RESTfulClient.execGET("http://"+
                    hueLampAdapter.DEFAULT_BRIDGE+"/api/fnp/lights/1/state
                    /on");
68              JSONObject obj = JSONParser.parseJSONObject(response.
                    getEntity().getContent());
69              assertEquals("{\"effect\":\"none\",\"on\":false}", obj.
                    toString());
70      }
71
72      @Test
73      public void blinkTest() throws Exception {
74              hueLampAdapter.setUp(null);
75              hueLampAdapter.setLampState(null, "blink");
76              HttpResponse response = RESTfulClient.execGET("http://"+
                    hueLampAdapter.DEFAULT_BRIDGE+"/api/fnp/lights/1/state
                    /on");
77              JSONObject obj = JSONParser.parseJSONObject(response.
                    getEntity().getContent());
78              assertEquals("{\"effect\":\"colorloop\",\"on\":true}", obj.
                    toString());
79      }
```

Listing 1.66.
../it.unibo.finalTask2018/test/it/unibo/test/impl/HueLampTest.java

I test mirano a verificare la correttezza della procedura di set-up (individuazione del bridge ed ottenimento di un username) e dei metodi per modificare lo stato della lampada.

8.4.2 Virtual Node-Robot test

Il componente che rappresenta il robot virtuale necessita di interagire con il server Node sul quale si trova l'implementazione utilizzata per il robot e per l'ambiente virtuale in cui si muove. Tale interazione è bidirezionale, dal momento che da un lato vengono inviati i comandi di movimento e dall'altro i dati relativi ai vari sonar.

I test devono pertanto verificare la comunicazione in entrambe le direzioni: per farlo si ricorre ad un oggetto mock che funga da server, così da poterne

controllare sia i dati inviati sia quello che viene effettivamente ricevuto.

```
28        private static QActor logger;
29        private static List<String> receivedMsgs;
30
31        @BeforeClass
32        public static void setUpBeforeClass() throws Exception {
33                MainCtxNodeRobotTest.initTheContext();
34                logger = waitForQActorToStart("nrtlogger");
35        }
36
37        @Test
38        public void tcpConnCmdEmissionTest() throws Exception {
39                Thread t = null;
40                try {
41                        receivedMsgs = new ArrayList<>();
42                        TCPServer server = new TCPServer(7777, s -> {
43                                receivedMsgs.add(s);
44                                return "";
45                        });
46                        t = server.runOnThread();
47
48                        clientTcp.initClientConn(logger, "localhost", 7777)
                              ;
49                        // usa clientTcp per inviare i dati nel giusto
                              formato (JSON-like)
50                        NodeRobot robot = new NodeRobot();
51                        robot.forward(logger);
52                        robot.backward(logger);
53                        robot.stop(logger);
54                        robot.left(logger);
55                        robot.right(logger);
56
57                        Thread.sleep(1000);
58
59                        assertEquals(5, receivedMsgs.size());
60                        assertEquals(";{\"arg\":-1,\"type\":\"moveForward
                              \"};", receivedMsgs.get(0));
61                        assertEquals(";{\"arg\":-1,\"type\":\"moveBackward
                              \"};", receivedMsgs.get(1));
62                        assertEquals(";{\"arg\":800,\"type\":\"alarm\"};",
                              receivedMsgs.get(2));
63                        assertEquals(";{\"arg\":800,\"type\":\"turnLeft\"};
                              ", receivedMsgs.get(3));
64                        assertEquals(";{\"arg\":800,\"type\":\"turnRight
                              \"};", receivedMsgs.get(4));
65                } finally {
66                        if (t != null && t.isAlive())
67                                t.interrupt();
```

```
68              }
69      }
70
71      @Test
72      public void tcpConnEventsReceptionTest() throws Exception {
73              Thread t = null;
74              try {
75                      TCPServer server = new TCPServer(7778, s -> s);
76                      t = server.runOnThread();
77
78                      clientTcp.initClientConn(logger, "localhost", 7778)
                            ;
79                      PrintWriter writer = clientTcp.getPrintWriter();
80
81                      writer.println(
82                                      ";{ \"type\": \"sonar-activated\", \"
                                            arg\": { \"sonarName\": \"sonar
                                            -1\", \"distance\": 1, \"axis\":
                                            \"x\" } };");
83                      writer.println(
84                                      ";{ \"type\": \"sonar-activated\", \"
                                            arg\": { \"sonarName\": \"sonar
                                            -2\", \"distance\": 2, \"axis\":
                                            \"x\" } };");
85                      writer.println(";{ \"type\": \"collision\", \"arg
                            \": { \"objectName\": \"wall\" } };");
86                      writer.flush();
87
88                      Thread.sleep(1000);
89
90                      assertTrue(isEventReceived(logger, "sonarSensor", "
                            sonar(sonar1,1)"));
91                      assertTrue(isEventReceived(logger, "sonarSensor", "
                            sonar(sonar2,2)"));
92                      assertTrue(isEventReceived(logger, "frontSonar", "
                            sonar(2)"));
93              } finally {
94                      if (t != null && t.isAlive())
95                              t.interrupt();
96              }
97      }
```

Listing 1.67.
../it.unibo.finalTask2018/test/it/unibo/test/impl/NodeRobotTest.java

8.4.3 Real Rasp-Robot test

Come illustrato nella Subsection 7.4, il robot reale che si trova sul RaspberryPi si rivolge ad un server scritto in Python che mostra all'esterno una visione di alto livello dei sensori e degli attuatori collegati alla board di Arduino.

Il server, opportunamente simulato a fini di test, deve ricevere sia i comandi di movimento del robot sia quelli che regolano il led presente su di esso, tutti espressi nel giusto formato; inoltre, deve rendere disponibili i dati del sonar frontale al relativo client Java, così che questi possa emetterli sotto forma di eventi `frontSonar`.

```
18      private static QActor realrobotrasp;
19      private static QActor ledagent;
20      private static QActor rrrlogger;
21
22      private static Process appl;
23      private static final int SRV_PORT = 6666;
24      private static List<String> receivedMsgs = new LinkedList<>();
25      private static Thread serverThread;
26      private static final String FRONT = "10.10";
27
28      @BeforeClass
29      public static void setUpBeforeClass() throws Exception {
30              // in alternativa avviare (python3):
31              // ..\raspberry\pyServer\server_mock_no_arduino.py
32              TCPServer server = new TCPServer(SRV_PORT, msg -> {
33                      receivedMsgs.add(msg);
34                      return FRONT;  // front sonar
35              });
36              serverThread = server.runOnThread();
37
38              System.out.println("Starting MainCtxAppl...");
39              appl = execMain(MainCtxAppl.class);
40              Thread.sleep(5000);
41
42              MainCtxRealRobotRasp.initTheContext();
43              realrobotrasp = waitForQActorToStart("realrobotrasp");
44              ledagent = waitForQActorToStart("ledagent");
45              rrrlogger = waitForQActorToStart("rrrlogger");
46              Thread.sleep(2000);
47      }
48
49      @AfterClass
50      public static void tearDownAfterClass() throws Exception {
51              stopProcess(appl);
52
53              if(serverThread != null && serverThread.isAlive())
54                      serverThread.interrupt();
55      }
```

```
56
57      @Test
58      public void moveRobotReceptionTest() throws Exception {
59              ledagent.emit("ctrlEvent", "ctrlEvent(robot,r1,w(0))");
60              Thread.sleep(1000);
61              ledagent.emit("ctrlEvent", "ctrlEvent(robot,r1,a(0))");
62              Thread.sleep(1000);
63
64              String wCmd = realrobotrasp.solveGoal("map(w(0),OUT)").
                    getTerm("OUT").toString();
65              String aCmd = realrobotrasp.solveGoal("map(a(0),OUT)").
                    getTerm("OUT").toString();
66              assertTrue(receivedMsgs.contains(wCmd));
67              assertTrue(receivedMsgs.contains(aCmd));
68      }
69
70      @Test
71      public void lightCmdReceptionTest() throws Exception {
72              realrobotrasp.emit("lightCmd", "lightCmd(on)");
73              Thread.sleep(1000);
74
75              String onCmd = ledagent.solveGoal("map(on,OUT)").getTerm("
                    OUT").toString();
76              assertTrue(receivedMsgs.contains(onCmd));
77      }
78
79      @Test
80      public void frontSonarEmissionTest() throws Exception {
81              // il server mock (TCPServer) risponde ad ogni comando
                    ricevuto inviando un dato del sonar
82              realrobotrasp.emit("lightCmd", "lightCmd(off)");
83              Thread.sleep(1000);
84              assertTrue(isEventReceived(rrrlogger, "frontSonar", "sonar(
                    " + FRONT + ")"));
85      }
```

Listing 1.68. ../it.unibo.raspRobot/test/RealRobotTest.java

8.4.4 Real temperature sensor test

Anche l'implementazione *Arduino-based* del servizio che fornisce la temperatura deve comunicare con un apposito server Python, il quale risponde ad ogni richiesta con il dato aggiornato sulla temperatura corrente. Alla ricezione di questi, `TemperatureAgent` deve emettere nel sistema eventi `temperature`.

```
20      private static QActor crslogger;
21      private static QActor tempagent;
22
```

```
23          private static final int SRV_PORT = 6667;
24          private static List<String> receivedMsgs = new LinkedList<>();
25          private static Thread serverThread;
26          private static Process thermoServer;
27          private static final String TEMP = "20.20";
28
29          private static void setUpMockServer() {
30                  TCPServer server = new TCPServer(SRV_PORT, req -> {
31                          receivedMsgs.add(req);
32                          return TEMP;
33                  });
34                  serverThread = server.runOnThread();
35          }
36
37          private static void setUpThermoServer() throws Exception {
38                  thermoServer = Runtime.getRuntime().exec("java -jar ../
                        ThermoServer.jar");
39          }
40
41          @BeforeClass
42          public static void setUpBeforeClass() throws Exception {
43                  setUpMockServer();
44  //      setUpThermoServer();
45                  MainCtxAppl.initTheContext();
46                  crslogger = waitForQActorToStart("crslogger");
47                  tempagent = waitForQActorToStart("temperatureagent");
48          }
49
50          @AfterClass
51          public static void tearDownAfterClass() throws Exception {
52                  stopProcess(thermoServer);
53                  if (serverThread != null && serverThread.isAlive())
54                          serverThread.interrupt();
55          }
56
57          @Test
58          public void temperatureEmissionTest() throws Exception {
59                  int period = Integer.parseInt(tempagent.solveGoal("
                        requestPeriod(P)").getVarValue("P").toString());
60                  Thread.sleep(period + 1000);
61                  if (serverThread != null)
62                          assertFalse(receivedMsgs.isEmpty());
63                  assertTrue(isEventReceived(crslogger, "temperature", "
                        temperature(" + TEMP + ")"));
64          }
```

Listing 1.69.
../it.unibo.finalTask2018/test/it/unibo/test/impl/TemperatureAgentTest.java

9 Authors

Photo of the authors