

Software Engineering process

Luca Bonfiglioli, Nicola Fava, Antonio Grasso

Alma Mater Studiorum – University of Bologna
viale Risorgimento 2, 40136 Bologna, Italy
`luca.bonfiglioli10@studio.unibo.it`
`nicola.fava@studio.unibo.it`
`antonio.grasso5@studio.unibo.it`

Table of Contents

Software Engineering process	1
<i>Luca Bonfiglioli, Nicola Fava, Antonio Grasso</i>	
1 Introduzione	3
2 Vision	3
3 Requisiti	4
4 Analisi dei requisiti	5
5 Analisi del problema	10
6 Progettazione	15
7 Implementazione	15
8 Autori	16

1 Introduzione

L'ingegneria diversifica le fasi di produzione del software delineando un flusso di lavoro (**workflow**) costituito da un insieme di passi: definizione dei requisiti, analisi dei requisiti, analisi del problema, progettazione della soluzione, implementazione della soluzione e collaudo.

La progettazione del software può seguire due approcci:

- **Approccio top-down**: si considera l'intero sistema software come un'unica entità e lo si scompone per ottenere più di un sotto-sistema o componente. Ogni sotto-sistema o componente viene considerato come un sistema e ulteriormente decomposto;
- **Approccio bottom-up**: si compongono componenti di più alto livello utilizzando componenti base o di più basso livello. Si continua a creare componenti di più alto livello finché il sistema desiderato non si evolve come un singolo componente.

I problemi possono essere affrontati utilizzando due differenti approcci:

- **Approccio olistico**: un sistema viene visto come un insieme che va oltre i sotto-sistemi o i componenti di cui è costituito;
- **Approccio riduzionistico**: non può essere sviluppato nessun sistema a meno che non si conoscano informazioni su di esso e sui componenti di cui si compone.

Occorre chiedersi se sia meglio tentare di risolvere un problema partendo dalle ipotesi tecnologiche (come possono essere ad esempio gli oggetti Java) o piuttosto seguire un approccio in cui l'analisi del problema precede la scelta della tecnologia più appropriata. Dopo aver completato l'analisi del problema è possibile imbattersi in un cosiddetto **abstraction gap**, che evidenzia un gap tra le tecnologie disponibili ed il problema che si deve risolvere.

2 Vision

L'obiettivo dell'analisi dei requisiti è capire cosa voglia il committente al fine di produrre, al termine dell'analisi, uno o più modelli nel modo più formale e pratico possibile.

Lo scopo principale della fase di analisi del problema è quello di capire il problema posto dai requisiti, le problematiche riguardanti il problema e i vincoli imposti dal problema o dal contesto. Il risultato dell'analisi del problema è l'architettura logica richiesta dal problema stesso.

All'inizio del processo di sviluppo del software non si considera nessuna ipotesi tecnologica (come ad esempio il paradigma di programmazione ad oggetti o il paradigma di programmazione funzionale).

3 Requisiti

Nella casa di una determinata città (per esempio Bologna), viene usato un `ddr` robot per pulire il pavimento di una stanza (`R-FloorClean`).

Il pavimento della stanza è un pavimento piatto di materiale solido ed è equipaggiato con due *sonars*, chiamati `sonar1` e `sonar2`, come mostrato in Figura 1 (`sonar1` è quello in alto). La posizione iniziale (`start-point`) del robot è rilevata da `sonar1`, mentre la posizione finale (`end-point`) da `sonar2`.

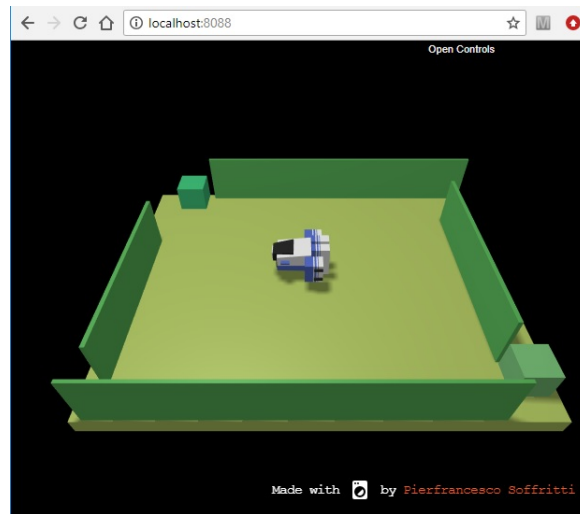


Fig. 1. Esempio di pavimento con il robot in ambiente simulato

Il robot lavora secondo le seguenti condizioni:

1. `R-Start`: un utente autorizzato (`authorized user`) ha inviato un comando `START` usando un'interfaccia GUI umana (`console`) in esecuzione su un normale PC oppure su uno smart device (`Android`).
2. `R-TempOk`: il valore di temperatura della città non è superiore ad un valore prefissato (per esempio 25° Celsius).
3. `R-TimeOk`: l'orario corrente è all'interno di un intervallo dato (per esempio fra le 7 e le 10 di mattina).

Mentre il robot è in movimento:

- un Led posto su di esso deve lampeggiare, se il robot è un `real` robot (`R-BlinkLed`);
- una Led Hue Lamp disponibile nella casa deve lampeggiare, se il robot è un `virtual` robot (`R-BlinkHue`);
- deve evitare gli ostacoli fissi (per esempio i mobili) presenti nella stanza (`R-AvoidFix`) e/o gli ostacoli mobili come palloni, gatti, ecc. (`R-AvoidMobile`).

Inoltre il robot deve interrompere la sua attività quando si verifica una delle seguenti condizioni:

1. **R-Stop**: un utente autorizzato (**authorized user**) ha inviato il comando di STOP utilizzando la **console**.
2. **R-TempKo**: il valore di temperatura della città diventa più alto del valore prefissato.
3. **R-TimeKo**: l'orario corrente non è più all'interno dell'intervallo dato.
4. **R-Obstacle**: il robot ha trovato un ostacolo che non è in grado di evitare.
5. **R-End**: il robot ha finito il suo lavoro.

Durante il suo funzionamento il robot può opzionalmente:

- **R-Map**: costruire una mappa del pavimento della stanza con la posizione degli ostacoli fissi. Una volta ottenuta, la mappa può essere utilizzata per definire un piano per un percorso (ottimo) dallo **start-point** all'**end-point**.

4 Analisi dei requisiti

Il sistema da modellare sarà, come esplicitato dai requisiti, eterogeneo e distribuito, in particolare composto da almeno due nodi: il nodo "Robot" e il nodo "PC/Android".

Per la modellazione si utilizza il linguaggio *QActor* in quanto adatto alla modellazione di sistemi distribuiti.

Il diagramma informale risultato dall'analisi dei requisiti è riportato in figura 2.

Il primo dei due nodi che si è modellati è il nodo "PC/Android" che si occupa di mostrare la GUI e di interagire direttamente con un utente umano, richiedendone l'autenticazione. Come da requisito **R-Start** l'interfaccia utente deve poter essere utilizzabile sia su PC che su un dispositivo **Android**. Tuttavia, essendo le funzioni che essa deve svolgere identiche in entrambi i casi, si sono rappresentati entrambi i nodi come un unico nodo. Su questo nodo esegue l'attore "GUI/Authenticator", che consente all'utente di autenticarsi e inviare i comandi di START e STOP al robot (**R-Start** e **R-Stop**).

Il secondo nodo che si è modellato è il nodo "Raspberry/PC", responsabile del controllo del robot. Esso può essere in esecuzione su un PC, nel caso del **virtual** robot, oppure su un Raspberry Pi nel caso del **real** robot. L'attore "Robot" si pone in attesa dei comandi inviati da "GUI/Authenticator" e riceve informazioni sull'ambiente esterno da un sensore di temperatura e da un timer (**R-TempOk**, **R-TimeOk**, **R-TempKo**, **R-TimeKo**). Durante l'esecuzione, se il robot è in movimento, l'attore "Robot" invia a "Led" e a "Hue Lamp" i comandi per l'accensione e lo spegnimento necessari a farli lampeggiare (**R-BlinkLed**, **R-BlinkHue**).

L'attore "Robot" si occupa inoltre di gestire la logica applicativa, che consiste, in seguito alla ricezione del comando START da parte dell'utente, nel prendere decisioni circa il movimento del robot all'interno della stanza – per il robot

reale – e all'interno dell'ambiente simulato – per il robot virtuale – tentando di evitare gli ostacoli fissi e mobili (**R-AvoidFix**, **R-AvoidMobile**) e costruendo una mappa (**R-Map**) dell'ambiente.

Inoltre, se l'attore "Robot" trova un ostacolo che non riesce ad evitare si deve fermare (**R-Obstacle**). Questa situazione si verifica quando il robot trova uno o più ostacoli che gli impediscono di avanzare in una direzione diversa da quella da cui è arrivato.

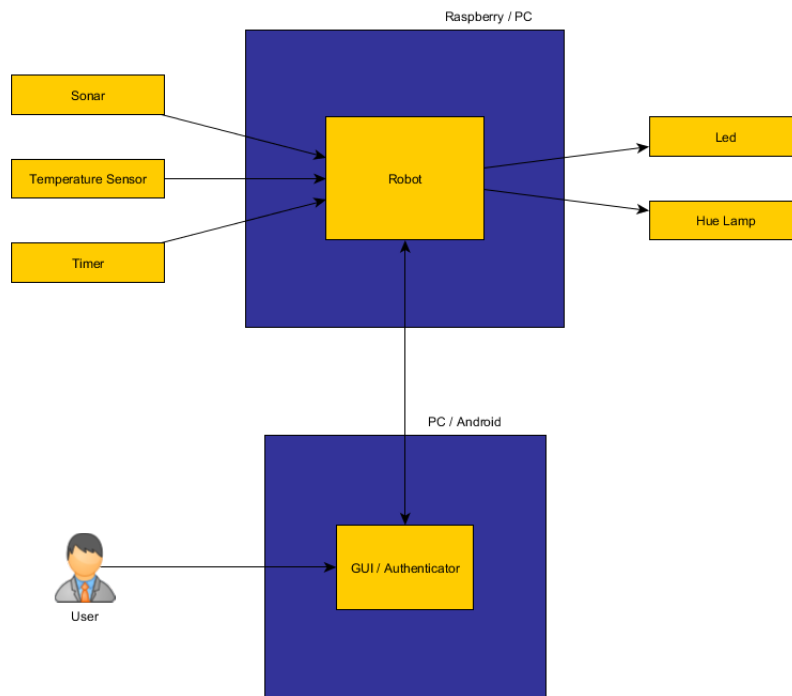


Fig. 2. Analisi dei requisiti informale

Vengono di seguito riportati i modelli formali risultati dall'analisi dei requisiti:

```

1 System systemRobot
2
3 Event robotCmd : robotCmd(X)
4 Event sensorEvent : sensorEvent(X)
5 Event outCmd : outCmd(X)
6
7 Context ctxRobot ip[host="localhost" port=5400]
8
9 QActor robot context ctxRobot {

```

```

10 Plan initial normal [
11     println("Robot started");
12     delay 2000
13 ]
14
15 switchTo waitForEvent
16
17 Plan waitForEvent [
18 ]
19 transition stopAfter 600000
20     whenEvent robotCmd -> handleEvent,
21     whenEvent sensorEvent -> handleEvent
22 finally repeatPlan
23
24 Plan handleEvent resumeLastPlan [
25     onEvent robotCmd : robotCmd(X) -> {
26         println("Robot receives event from user");
27         emit outCmd : outCmd(X)
28     };
29     onEvent sensorEvent : sensorEvent(sonar1) -> println("
Robot receives event from sonar1");
30     onEvent sensorEvent : sensorEvent(sonar2) -> println("
Robot receives event from sonar2");
31     onEvent sensorEvent : sensorEvent(temp) -> println("Robot
receives event from temperature sensor");
32     onEvent sensorEvent : sensorEvent(timer) -> println("
Robot receives event from timer sensor");
33     printCurrentEvent
34 ]
35 }
36
37 QActor sonarsensor1 context ctxRobot {
38     Plan initial normal[
39         println("Sonar1 started");
40         delay 3000
41     ]
42     switchTo emitEvents
43
44     Plan emitEvents[
45         emit sensorEvent : sensorEvent(sonar1)
46     ]
47 }
48
49 QActor sonarsensor2 context ctxRobot {
50     Plan initial normal[
51         println("Sonar2 started");
52         delay 3500
53     ]
54     switchTo emitEvents
55

```

```

56     Plan emitEvents[
57         emit sensorEvent : sensorEvent(sonar2)
58     ]
59 }
60
61 QActor temperaturesensor context ctxRobot {
62     Plan initial normal[
63         println("Temperature sensor started");
64         delay 4000
65     ]
66     switchTo emitEvents
67
68     Plan emitEvents[
69         emit sensorEvent : sensorEvent(temp)
70     ]
71 }
72
73 QActor timersensor context ctxRobot {
74     Plan initial normal[
75         println("Timer sensor started");
76         delay 4500
77     ]
78     switchTo emitEvents
79
80     Plan emitEvents[
81         emit sensorEvent : sensorEvent(timer)
82     ]
83 }
84
85 QActor led context ctxRobot {
86     Plan initial normal[
87         println("Led started")
88     ]
89     switchTo waitForEvent
90
91     Plan waitForEvent[]
92     transition stopAfter 600000
93     whenEvent outCmd -> handleEvent
94     finally repeatPlan
95
96     Plan handleEvent resumeLastPlan [
97         println("Led receives event");
98         printCurrentEvent
99     ]
100 }
101
102 QActor huelamp context ctxRobot {
103     Plan initial normal[
104         println("Hue Lamp started")
105     ]

```



```

106     switchTo waitForEvent
107
108     Plan waitForEvent []
109     transition stopAfter 600000
110     whenEvent outCmd -> handleEvent
111     finally repeatPlan
112
113     Plan handleEvent resumeLastPlan [
114         println("Hue Lamp receives event");
115         printCurrentEvent
116     ]
117 }

```

Listato 1.1. code/reqAnalysisRobot.qa

```

1  System systemRobot
2
3  Dispatch userCmd : userCmd(X)
4  Event robotCmd : robotCmd(X)
5
6  Context ctxRobot ip[host="localhost" port=5400] -standalone
7  Context ctxUser ip[host="localhost" port=5500]
8
9  QActor gui context ctxUser {
10     Plan initial normal [
11         println("Gui started")
12     ]
13     switchTo waitForMsg
14
15     Plan waitForMsg [
16     ]
17     transition stopAfter 600000
18     whenMsg userCmd -> handleMsg
19     finally repeatPlan
20
21     Plan handleMsg resumeLastPlan [
22         println("Gui receives user message - User pressed button"
23         );
24         onMsg userCmd : userCmd(X) -> emit robotCmd : robotCmd(X)
25     ]
26 }
27
28 QActor user context ctxUser {
29     Plan initial normal [
30         println("User started")
31     ]
32     switchTo sendMsg
33
34     Plan sendMsg[
35         println("User send messages");

```

```

35     forward gui -m userCmd : userCmd(X);
36     forward gui -m userCmd : userCmd(Y)
37 ]
38 }

```

Listato 1.2. code/reqAnalysisUser.qa

5 Analisi del problema

Il primo problema che sorge è quello di stabilire quale nodo si occuperà di autenticare l'utente. Una possibilità è che sia sul nodo del robot: in questo caso il robot potrebbe non disporre delle adeguate risorse computazionali per gestire il processo di autenticazione, tuttavia questo garantirebbe maggiore sicurezza. Un'altra possibilità è che l'autenticazione sia su un nodo diverso rispetto a quello del robot: ciò consente di non utilizzare le risorse computazionali del robot richiedendo però maggiori accortezze sulla sicurezza. In quest'ultimo caso l'autenticazione potrebbe essere gestita dal nodo dell'utente oppure da un nodo distinto, il quale comporterebbe costi maggiori.

Un altro problema è quello dell'interfaccia GUI, che deve poter eseguire su dispositivi eterogenei. A tal proposito, una possibilità sarebbe creare client nativi per ogni piattaforma con costi elevati oppure più semplicemente utilizzare una pagina web.

La comunicazione tra utente e robot tramite GUI può avvenire via messaggi o via eventi. La comunicazione ad eventi permette di disaccoppiare GUI e robot, consentendo di utilizzare un'unica GUI per comunicare con diversi robot. Utilizzando gli eventi può essere adottato un approccio **event-based** o un approccio **event-driven**. Nell'approccio **event-based** il robot non sarebbe sempre sensibile agli eventi, potendone perdere alcuni. Al contrario, nell'approccio **event-driven** il robot sarebbe sempre sensibile agli eventi perdendo tuttavia reattività.

```

1  System systemRobot
2
3  Event robotCmd : robotCmd(X)
4  Event sensorEvent : sensorEvent(X)
5  Event outCmd : outCmd(X)
6
7  Context ctxProbRobot ip[host="localhost" port=5400]
8
9  QActor robot context ctxProbRobot {
10     Rules {
11         limitTemperatureValue(25).
12         minTime(7).
13         maxTime(10).
14         currentTempValue(0).
15         currentTimeValue(0).
16         evalTemp:-

```

```

17         limitTemperatureValue(MAX),
18         currentTempValue(VALUE),
19         eval(ge, MAX, VALUE).
20     evalTime:-
21         minTime(MIN),
22         maxTime(MAX),
23         currentTimeValue(VALUE),
24         eval(ge, VALUE, MIN),
25         eval(ge, MAX, VALUE).
26     startRequirementsOk :- evalTemp, evalTime.
27     map.
28 }
29
30 Plan initial normal [
31     println("Robot started");
32     delay 2000
33 ]
34
35 switchTo waitForEvent
36
37 Plan waitForEvent [
38 ]
39 transition stopAfter 600000
40     whenEvent robotCmd -> handleEvent,
41     whenEvent sensorEvent -> handleEvent
42 finally repeatPlan
43
44 Plan handleEvent resumeLastPlan [
45     onEvent robotCmd : robotCmd(cmdstart) -> {
46         [ !? startRequirementsOk ] {
47             println("Robot start");
48             emit outCmd : outCmd(startblinking);
49             [ !? map ]
50             println("Il robot segue il percorso ottimo")
51         else
52             println("Mentre il robot è in azione deve costruire
53             la mappa")
54         }
55     };
56     onEvent robotCmd : robotCmd(cmdstop) -> {
57         println("Robot stop from user");
58         emit outCmd : outCmd(stopblinking)
59     };
60     onEvent sensorEvent : sensorEvent(sonar1) -> println("
61     Robot receives event from sonar1");
62     onEvent sensorEvent : sensorEvent(sonar2) -> {
63         emit robotCmd : robotCmd(cmdstop); // il robot ha
64         finito
65     }
66     [ !? map ]

```

```

63         addRule map // se non c'era una mappa precedente
viene creata
64     };
65     onEvent sensorEvent : sensorEvent(onboardsonar) -> {
66         //the robot stops for a while and then retries
67         println("Robot stop");
68         // se è la prima volta che incontra un ostacolo:
69         delay 500;
70         println("Il robot prova di nuovo ad andare avanti");
71         // se invece è la seconda volta che incontra lo stesso
ostacolo:
72         println("Il robot prova ad andare a destra o a sinistra
");
73         // se non si può andare né a destra né a sinistra
allora il robot si ferma:
74         emit robotCmd : robotCmd(cmdstop)
75     };
76     onEvent sensorEvent : sensorEvent(temp(VALUE)) ->
ReplaceRule currentTempValue(X) with currentTempValue(
VALUE);
77     onEvent sensorEvent : sensorEvent(temp(X)) -> {
78         [ not !? evalTemp ] {
79             println("Robot stop from temperature sensor");
80             emit outCmd : outCmd(stopblinking)
81         }
82     };
83     onEvent sensorEvent : sensorEvent(timer(VALUE)) ->
ReplaceRule currentTimeValue(X) with currentTimeValue(
VALUE);
84     onEvent sensorEvent : sensorEvent(timer(X)) -> {
85         [ not !? evalTime ] {
86             println("Robot stop from time sensor");
87             emit outCmd : outCmd(stopblinking)
88         }
89     };
90     printCurrentEvent
91 ]
92 }
93
94 QActor sonarsensor1 context ctxProbRobot {
95     Plan initial normal[
96         println("Sonar1 started");
97         delay 3000
98     ]
99     switchTo emitEvents
100
101     Plan emitEvents[
102         emit sensorEvent : sensorEvent(sonar1)
103     ]
104 }

```

```

105
106 QActor sonarsensor2 context ctxProbRobot {
107     Plan initial normal[
108         println("Sonar2 started");
109         delay 3500
110     ]
111     switchTo emitEvents
112
113     Plan emitEvents[
114         emit sensorEvent : sensorEvent(sonar2)
115     ]
116 }
117
118 QActor sonarrobot context ctxProbRobot {
119     Plan initial normal[
120         println("Sonar on board started");
121         delay 5000
122     ]
123     switchTo emitEvents
124
125     Plan emitEvents[
126         emit sensorEvent : sensorEvent(onboardsonar)
127     ]
128 }
129
130 QActor temperaturesensor context ctxProbRobot {
131     Plan initial normal[
132         println("Temperature sensor started");
133         delay 4000
134     ]
135     switchTo emitEvents
136
137     Plan emitEvents[
138         emit sensorEvent : sensorEvent(temp(20));
139         delay 2000;
140         emit sensorEvent : sensorEvent(temp(30))
141     ]
142 }
143
144 QActor timersensor context ctxProbRobot {
145     Plan initial normal[
146         println("Timer sensor started");
147         delay 4500
148     ]
149     switchTo emitEvents
150
151     Plan emitEvents[
152         emit sensorEvent : sensorEvent(timer(9));
153         delay 2000;
154         emit sensorEvent : sensorEvent(timer(12))

```

```

155 ]
156 }
157
158 QActor led context ctxProbRobot {
159   Plan initial normal [
160     println("Led started")
161   ]
162   switchTo waitForEvent
163
164   Plan waitForEvent []
165   transition stopAfter 600000
166     whenEvent outCmd -> handleEvent
167   finally repeatPlan
168
169   Plan handleEvent resumeLastPlan [
170     onEvent outCmd : outCmd(startblinking) -> println("Led
171     start blinking");
172     onEvent outCmd : outCmd(stopblinking) -> println("Led
173     stop blinking")
174   ]
175 }
176
177 QActor huelamp context ctxProbRobot {
178   Plan initial normal [
179     println("Hue Lamp started")
180   ]
181   switchTo waitForEvent
182
183   Plan waitForEvent []
184   transition stopAfter 600000
185     whenEvent outCmd -> handleEvent
186   finally repeatPlan
187
188   Plan handleEvent resumeLastPlan [
189     onEvent outCmd : outCmd(startblinking) -> println("Hue
190     Lamp start blinking");
191     onEvent outCmd : outCmd(stopblinking) -> println("Hue
192     Lamp stop blinking")
193   ]
194 }

```

Listato 1.3. code/probAnalysisRobot.qa

```

1 System systemRobot
2
3 // payload: cmdstart o cmdstop
4 Dispatch userCmd : userCmd(X)
5 Event robotCmd : robotCmd(X)
6

```

```

7 Context ctxProbRobot ip[host="localhost" port=5400] -
  standalone
8 Context ctxProbUser ip[host="localhost" port=5500]
9
10 QActor gui context ctxProbUser {
11   Plan initial normal [
12     println("Gui started")
13   ]
14   switchTo waitForMsg
15
16   Plan waitForMsg [
17   ]
18   transition stopAfter 600000
19     whenMsg userCmd -> handleMsg
20   finally repeatPlan
21
22   Plan handleMsg resumeLastPlan [
23     println("Gui receives user message");
24     onMsg userCmd : userCmd(X) -> emit robotCmd : robotCmd(X)
25   ]
26 }
27
28 QActor user context ctxProbUser {
29   Rules {
30     isUserAuthenticated.
31   }
32
33   Plan initial normal [
34     println("User started")
35   ]
36   switchTo sendMsg
37
38   Plan sendMsg[
39     println("User send messages");
40     [ !? isUserAuthenticated ]
41     forward gui -m userCmd : userCmd(cmdstart);
42     delay 2000;
43     [ !? isUserAuthenticated ]
44     forward gui -m userCmd : userCmd(cmdstop)
45   ]
46 }




```

Listato 1.4. code/probAnalysisUser.qa

6 Progettazione

7 Implementazione

8 Autori

Foto degli autori		
		
Luca Bonfiglioli	Nicola Fava	Antonio Grasso