Final task
LABORATORIO DI SISTEMI SOFTWARE

Pecci Federica Varini Chiara

December 2018

1 Requirements Analysis

Questa sezione illustra tutti i passi necessari per svolgere la fase di analisi dei requisiti. Prima di tutto viene chiarito il significato di tutti i termini e concetti riportati nel file "Tema finale" (dato dal committente), cosicché l'analista abbia una chiara comprensione di ciò che il cliente si aspetta che il sistema faccia. Dopodiché è presentata una formalizzazione del sistema utilizzando i QActor: viene rappresentanto un modello per ogni componente del sistema.

1.1 Requirements Description

La descrizione dei requisiti comprende lo svisceramento delle seguenti domande:

1. **Di che sistema necessita il committente?**
   Il committente necessita di un sistema composto da un robot che deve esplorare tutta la hall di un aeroporto (R-explore) e scattare una foto (R-takePhoto) quando giunge in prossimità di una valigia. Ogni foto viene successivamente inviata ad un'altra parte del sistema denominata console (R-sendPhoto). Nel caso in cui la valigia risulti pericolosa un secondo robot deve provvedere al disinnescamento della bomba (R-reachBag). Si tratta dunque di un sistema distribuito eterogeneo.

2. **Da quanti componenti è composto il sistema?**
   Il sistema è composto da 2 componenti: 1 console e 1 robot. Anche se il committente ipotizza l'utilizzo di 2 robot fisici differenti, uno per l'esplorazione ed uno per la gestione della valigia bomba, questi possono essere modellati come due comportamenti diversi dello stesso robot fisico.

3. **Qual è il compito della console?**
   Il compito della console è interpretare i comandi ricevuti dall'operatore e, nel caso in cui siano validi, inviarli al robot. Inoltre la console deve riuscire a comunicare con i robot: ricevere le foto e inviare comandi.

4. **Che cosa si intende per ostacolo?**
   Gli ostacoli modellati nel sistema sono:

   - valige: lasciate dai passeggeri nella stanza al momento dell'evacuazione;
   - muri della stanza.

5. **Quando il robot può partire con l'esplorazione?**
   Il robot può partire con l'esplorazione quando le due condizioni del sistema sono verificate, ossia il robot riceve un comando di inizio perlustrazione e la temperatura della stanza è inferiore ad un certa soglia.

6. **Che cosa si intende per esplorazione autonoma di un robot?**
   Per esplorazione autonoma di un robot si intende la capacità di perlustrare interamente una stanza con ostacoli fissi (valigie e muri) ed una superficie piana. Durante questa fase il robot deve far blinkare un led posto su di esso.

7. **Quando il robot si deve fermare?**
   Il robot si deve fermare in 3 casi:
   - quando riceve un comando di stop (RstopExplore) dalla console;
   - quando si trova in prossima di una valigia non ancora esaminata (RstopAtBag);
   - quando la temperatura della hall supera la soglia fissata
8. **Quando il robot deve tornare alla base?**
   Il robot deve tornare alla base quando riceve il comando RbackHomeSinceBomb o RbackHome.
9. **Cosa fa il robot quando incontra un ostacolo?**
   Quando il robot incontra un ostacolo, se è una valigia si ferma, fa la foto, la manda alla console e aspetta un comando dalla console, invece, se è un muro, il robot cambia direzione e continua l'esplorazione.
10. **Cosa fa il robot una volta terminata l'ispezione della hall?**
    Il robot ha controllato tutta la hall senza trovare una valigia sospetta allora torna al suo punto di partenza.
11. **Quali informazioni deve conoscere la console sul robot?**
    La console deve avere delle informazioni riguardanti lo stato del robot per sapere se è in stato di esplorazione, se è fermo o se si trova nella posizione iniziale. Inoltre, deve poter ricevere le foto dei bagagli inviati dal robot e memorizzare le relative informazioni (orario e posizione del robot al momento dello scatto della foto).

1.2 QActor formalisation

Una formalizzazione di quanto descritto nella sottosezione precedente la si può ottenere usando il linguaggio dei QActor. In particolare, si è realizzato un sistema composto da due attori (una console ed un robot) che operano nello stesso contesto.

```
System ddrSys

Context ctx ip [ host= "localhost" port=8078] -g green

QActor console context ctx {
  Plan init normal [
    println( "console initialised" )
  ]
}

QActor robot context ctx {
  Plan init normal [
    println( "robot initialised" )
  ]
}
```

1.3 Unit Tests

//TODO

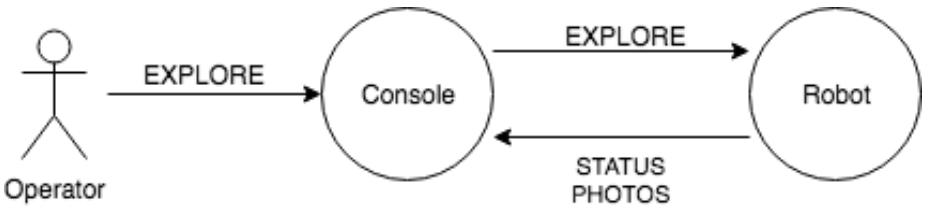

Figure 1: First and generic representation of the system.

2 Problem analysis

Le problematiche iniziali emerse dall'analisi dei requisiti e riguardanti i componenti sono:

1. **distribuzione**: il robot e la console sono fisicamente in due posti diversi, quindi il sistema deve essere distribuito;
2. **eterogeneità**: il robot e la console potrebbero utilizzare tecnologie diverse, quindi il sistema deve essere eterogeneo;
3. **interazione**: trattandosi di un sistema distribuito eterogeneo per l'interazione tre le entità si devono utilizzare il message passing e gli eventi;
4. **coordinazione**: il robot e la console devono coordinarsi tra loro, quindi è opportuno stabilire una policy per definire quando e come determinate azioni devono verificarsi.

Nella fig. 1 è raffigurata la prima modellazione, ad un alto livello di astrazione, del sistema. In questo modello è possibile osservare che vi sono 3 entità: operatore, console e robot. Essendo l'operatore un'entità esterna al sistema e considerando che esso rappresenta una mera entità il cui compito è inviare comandi alla console, nelle modellazioni successive esso verrà trascurato.

Successivamente, analizzando più nel dettaglio il problema sono emerse le seguenti problematiche:

1. **Come riceve i comandi la console?**
   In questo sistema, l'interazione tra l'operatore e la console può essere modellata a procedure call: l'operatore preme un pulsante su un'interfaccia grafica (come richiesto dal cliente), la console elabora il comando verificando che esso sia valido e chiama infine la procedura adatta alla sua gestione. Essendo tutto gestito internamente al componente console, risulta inutile mettere in campo interazioni di tipo message-passing o ad eventi.
2. **Come interagiscono console e robot?**
   Essendo un sistema distribuito la console e il robot interagiscono attraverso lo scambio di messaggi. I messaggi scambiati tra le due entità sono formalizzti di seguito.

```
//Message from console to robot
Dispatch explore: explore(X)
Dispatch stopExplore: stopExplore(X)
Dispatch backHome: backHome(X)
Dispatch continueExplore: continueExplore(X)
Dispatch backHomeSinceBomb: backHomeSinceBomb(X)
Dispatch continueExploreAfterPhoto: continueExploreAfterPhoto(
    X)

//Message from robot to console
Dispatch sendPhoto: sendPhoto(X)

//Message from robot to robot
Dispatch reachBag: reachBag(X)


```

3. **Come fa la console ad avere le informazioni riguardanti lo stato del robot?**
   Ogni volta che il robot cambia stato emette un evento, il quale conterrà le informazioni aggiornate riguardanti il nuovo stato: tale evento verrà percepito dalla console. L'evento emesso è formalizzato di seguito.

```
//robot state update event
Event consoleUpdate: consoleUpdate(X)
```

4. **Come viene percepito il cambiamento di temperatura della stanza?**
   Il cambiamento della temperatura nella stanza può essere modellato come un evento del sistema. In particolare, sarà possibile conoscere la temperatura della stanza utilizzando un microservizio, quindi non sarà necessario che vi sia un sensore di temperatura posto sul robot. L'evento emesso è formalizzato di seguito.

```
//temperature change event
Event tempOk: tempOk(X)
```

5. **Quanti eventi vengono emessi quando cambia la temperatura?**
   Ogni volta che la temperatura cambia in modo consistente (ad esempio di circa 1°) viene emesso un evento con il nuovo valore della temperatura.

6. **Chi gestisce l'evento del cambiamento della temperatura?**
   Ogni volta che la temperatura supera una certa soglia si scatena un evento che è percepito e gestito dalla console in quanto, fra i componenti, è quello scelto per incapsulare la logica di gestione dell'intero sistema. Di conseguenza non si ritiene opportuno che questo evento sia percepito dal robot.

7. **Come fa il robot ad esplorare la stanza?**
   La stanza viene esplorata in maniera proattiva dal robot, quest'ultimo costruisce progressivamente una mappa della stanza riportando su di essa ostacoli fissi, ossia valigie, e muri con relative posizioni (e dimensioni nel caso si tratta di valigie) .

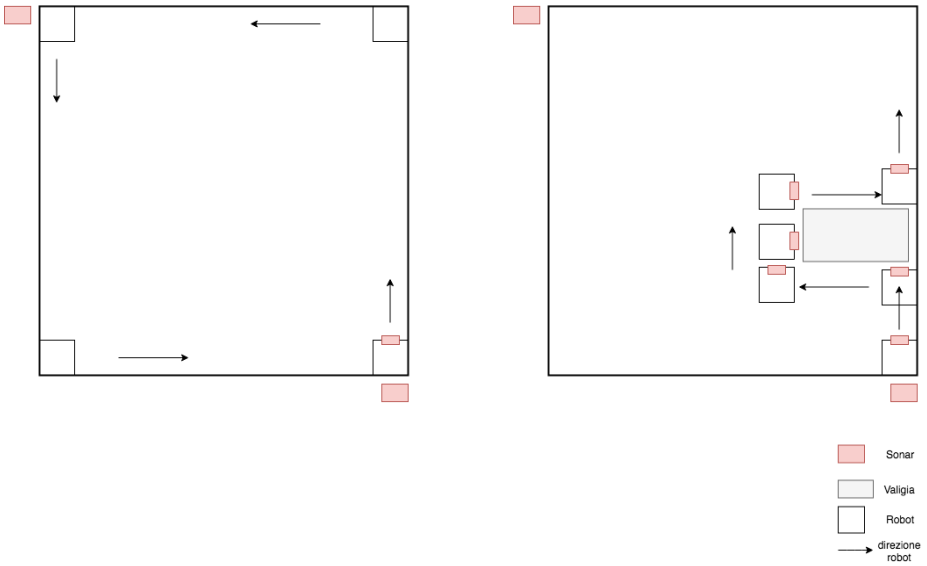

Figure 2: Robot exploration considering obstacoles.

8. **Come vengono gestiti gli ostacoli durante l'esplorazione?**
   La logica di gestione degli ostacoli durante l'esplorazione è incapsulata nel robot. Un esempio potrebbe essere: se l'ostacolo incontrato dal robot è un muro, esso cambia direzione e procede con l'eplorazione. Un altro esempio potrebbe essere: se l'ostacolo incontrato dal robot è una valigia, il robot scatta una foto di essa e la invia alla console poi, ne calcola le dimensioni compiendo un giro di 360° intorno ad esso oppure girandogli attorno sino a quando non incontra un muro che gli impiedisce di completare il giro. Infine, il robot salva la posizione e le dimensioni dell'ostacolo sulla mappa, torna nella posizione iniziale per poi riprendere l'esplorazione (Nella fig. 2, l'immagine a sinistra rappresenta l'esplorazione della stanza considerando solo i muri come ostacoli, invece quella a destra rappresenta l'esplorazione considerando muri e valigie).

9. **Come fa il robot a riconoscere un ostacolo?**
   Come già specificato, gli ostacoli possono essere di 3 tipi:

   - muro: l'ostacolo è un muro quando si attiva il sonar del robot ed è attivo anche uno dei due sonar presenti negli angoli opposti della stanza;
   - valigia non analizzata: l'ostacolo è una valigia (e non un muro) in quanto non sono attivi i sonar posti negli angoli opposti della stanza ma solamente quello posto sul robot. L'ostacolo non è ancora stato analizzato in quanto la sua posizione e le dimensioni non sono ancora state salvate sulla mappa del robot;

- valigia già analizzata: come nel caso precedente l'ostacolo è una valigia ed il robot apprende questa informazione consultando direttamente la mappa che ha a disposizione.

10. **Da quale angolazione il robot scatta la foto alla valigia?**
    Il robot scatta la foto alla valigia esattamente dall'angolazione in cui esso si trova rispetto all'ostacolo nel momento in cui giunge in sua prossimità. Infatti, l'angolazione della foto risulta ininfluente ai fini della valutazione della presenza o meno del bagaglio, poiché si suppone che il tool utlizzato dalla console esamini il bagaglio con una tecnologia a infrarossi. L'unica condizione necessaria per scattare una foto idonea per la valutazione eseguita dal tool è che il bagaglio sia fotografato per intero.

11. **Come fa il robot a tornare nella posizione iniziale?**
    La prima cella che il robot memorizza nella propria mappa è la sua posizione iniziale, quindi basterà che esso percorra un qualunque tragitto dalla sua posizione attuale alla prima cella memorizzata della mappa per far sì che torni nella sua posizione iniziale.

12. **In caso di valigia sospetta, come fa il robot ad andare a prendere la valigia indicata e portarla nella posizione iniziale?**
    La valigia sospetta sarà l'ultimo ostacolo memorizzato nella mappa, in quanto quando questo viene rilevato la fase di esplorazione si sospende. Quindi, per prendere la valigia indicata e portarla nella posizione iniziale, è necessario che il robot percorra un qualunque tragitto dalla sua posizione all'ultima cella memorizzata.

13. **Durante la fase di esplorazione come fa il robot a ricevere i comandi di stop esplorazione e di back home?**
    Il robot dovrà avere una natura proattiva per gestire autonomamente l'esplorazione della hall e un natura reattiva per ricevere e rispondere prontamente ai comandi di stop esplorazione. I comandi di esplorazione, stop esplorazione e back home sono gestiti tutti come messaggi.

2.1 Test Plan

Nel Test Plan bisogna verificare che:

- quando l'utente spinge il pulsante di startExploration e la temperaura della stanza è inferiore ad una soglia fissata la console invia il messaggio di explore al robot;

- ogni volta che la console percepisce l'evento del cambio di temperatura, verifica se è ancora adeguata e se è troppo alta invia al robot il messaggio di backHome;

- quando il robot riceve il messaggio di explore inizia ad esplorare tutta la stanza;

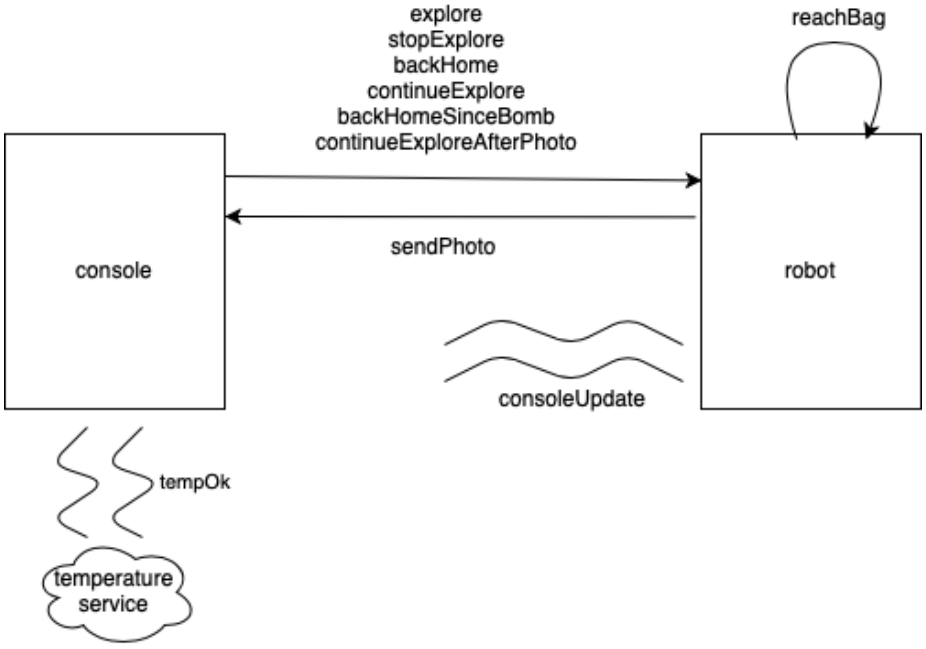

Figure 3: System logic architecture.

- quando il robot riceve il messaggio di stopExplore si ferma nel punto in cui si trova;
- quando il robot riceve, per volontà dell'operatore, il messaggio di backHome torna nella sua posizione iniziale;
- ogni volta che il robot incontra una valigia non ancora analizzata si ferma, gli scatta una foto e la spedisce con un messaggio di sendPhoto. Infine, gira attorno all'ostacolo per determinarne dimensioni e posizione, torna alla base e poi riprende l'esplorazione;
- quando il robot riceve il messaggio di continueExplore riprende l'esplorazione.

In base all'analisi del problema, si è derivata l'architettura logica di figura 3, la quale può essere formalizzata attraverso i QActor come nel codice riportato di seguito.

```
System ddrSys

//robot state update event
Event consoleUpdate: consoleUpdate(X)

//temperature change event
Event tempOk: tempOk(X)

//Message from console to robot
```

```
Dispatch explore: explore(X)
Dispatch stopExplore: stopExplore(X)
Dispatch backHome: backHome(X)
Dispatch continueExplore: continueExplore(X)
Dispatch backHomeSinceBomb: backHomeSinceBomb(X)
Dispatch continueExploreAfterPhoto: continueExploreAfterPhoto(X)

//Message from robot to console
Dispatch sendPhoto: sendPhoto(X)

//Message from robot to robot
Dispatch reachBag: reachBag(X)


Context ctx ip [host="localhost" port=8078] -g cyan

QActor console context ctx {

  Plan init normal [
    println("Console intialized")
  ]
}

QActor robot context ctx {
  Plan init normal [

    println("Robot intialized")
  ]
}
```

3 System model

4 Coding

5 Product BackLog

Sprint 0

30/12/2018
In questo sprint si sono definiti in modo formale i requisiti del sistema e si è analizzato in dettaglio il problema. Dall'analisi del problema è emersa l'architettura logica del sistema che è stata formalizzata utilizzando i QActor. (file *code/LogicArchitecture.qa.tex*)