Final Task LSS

Diego Pergolini
Luca Passeri

20/02/2019

Sommario

1 Introduzione

Per sviluppare il task propostoci dal committente la nostra software house si propone,di utilizzare una metodologia agile seguendo un approccio top-down model-driven. Adottando una metodologia Scrum-like tutto il lavoro sarà diviso in sprint, al termine di ognuno dei quali verrà compiuta una Sprint Review, in cui il team si rivolgerà al committente mostrando il lavoro fatto ed intervistandolo su requisiti non ancora studiati perché non chiaramente specificati negli sprint precedenti. Al termine di ogni sprint l'obbiettivo sarà quello di consegnare al client un prodotto che sia già di valore, pur non essendo completo in tutte le sue parti. Per questo motivo in ogni sprint si effettueranno tutte le fasi di un processo di sviluppo software, affinando progressivamente il risultato, migliorando l'esistente ed affrontando le problematiche non ancora esaminate. E' quindi evidente che la realizzazione del software seguirà un approccio incrementale con andatura monotona crescente, dove l'idea è quella di promuovere l'estendibilità ed il cambiamento senza impattare pesantemente ciò che è stato precedentemente realizzato. Perciò, per ogni sprint, nella prima fase si procederà con un'attenta analisi dei requisiti. Obbiettivo di questa fase sarà identificare con chiarezza tutti i requisiti del sistema, tracciandoli in maniera univoca e per ognuno di essi specificare entità messe in gioco ed interazioni fra di esse. Come valore aggiunto, si

intende, infine, formalizzare tutti i requisiti, in modo da ottenere una solida base di partenza che definerà il modello del sistema. Dovremo inoltre prevedere un preciso e puntuale piano di testing. Terminata la fase di analisi dei requisiti andranno sviscerate tutte le problematiche relative alla progettazione del sistema, senza scendere in dettagli implementativi, cercando di essere il più technology independent possibile ma al contempo consapevoli dei vincoli e dei pregi legati ad ogni tecnologia. L'output di questa fase sarà l'architettura logica del sistema, che se fatta correttamente, rimarrà invariata durante tutte le fasi successive.

2 Sprint 1

2.1 Sprint Planning

Essendo questo lo sprint iniziale, ci proponiamo di effettuare un'attenta analisi dei requisiti con uno sguardo generale su tutto il sistema, una volta effettuata questa analisi iniziale potrà quindi confrontarsi con il committente per chiarire eventuali aspetti ambigui.
In questa fase dovremmo innanzitutto individuare la struttura e le interazioni di massima del sistema. Nella fase di analisi dei requisiti emergeranno quali i sono i componenti del sistema per il committente, mentre nella fase di analisi del problema potranno emergere nuovi componenti di interesse per la futura progettazione.
In questo sprint l'analisi dei requisiti costituirà l'impegno più oneroso, in quanto vogliamo far emergere il prima possibile eventuali ambiguità da chiarire con il committente. Un sottoinsieme dei requisiti emersi, ritenuti prioritari, verrà esaminato fin da subito con l'analisi del problema, così da fornire un modello formale ed eseguibile da mostrare al cliente.
Negli sprint successivi analizzeremo più in profondità altri aspetti più applicativi legati al sistema, infatti in questa fase vogliamo costruire soltanto uno scheletro di base del software.

2.2 Analisi dei Requisiti

In questa sezione procederemo analizzando requisito per requisito il problema, individuando le entità in gioco (struttura), i rapporti fra di esse (interazione) e di conseguenza il loro comportamento.

Utilizzeremo il concetto della tracciabilità dei requisiti, in modo da identificare chiaramente lungo il tutto processo quali sono gli obbiettivi che si stanno perseguendo e per constatare alla fine quali requisiti sono stati soddisfatti e quali no ed in che parte del progetto. In prima analisi è bene identificare chiaramente quali sono le entità coinvolte nel sistema, definendo al contempo un glossario dei termini.
Il sistema è composto in prima battuta da una console, un ddr robot dedicato all'esplorazione ed uno dedicato alla rimozione dell'ordigno.

2.2.1 Componenti identificati in fase di analisi requisiti

Console
Che cos'è la console per il committente?
La console è rappresentata da un'interfaccia grafica, attraverso la quale l'operatore può interagire con il sistema.
Che cosa fa la console?
La console deve permettere di interagire con il robot esploratore inviandogli un comando per far partire l'esplorazione.
Attraverso la console l'utente può stoppare l'esplorazione per poi scegliere di continuarla oppure di farlo tornare alla base.
In caso di rilevamento di un ordigno la console deve farne notifica all'utente, salvare la foto ed inviare al robot il comando per tornare alla posizione iniziale.
Infine deve consentire l'invio di un comando per far si che un altro robot possa recuperare l'ordigno.
Robot Discovery
Che cos'è il robot discovery per il committente?
Si tratta di un ddr robot che si occupa dell'esplorazione dell'ambiente al fine di rilevare la presenza di eventuali ordigni. Il robot per il committente è un dispositivo fisico in grado di muoversi nell'ambiente grazie a delle ruote motrici,fornito di un led e dotato di un sonar posto frontalmente per individuare ostacoli.
Che cosa fa il robot discovery?
Esplora l'ambiente con il compito di individuare ogni valigia presente sul pavimento della stanza.
Permette di essere avviato e/o stoppato remotamente.
Deve far lampeggiare un led ed inviare informazione sullo stato suo e dell'esplorazione.
In prossimità di una valigia deve fermarsi, scattare una foto ed inviarla

all'operatore.
Robot Retriever
Che cos'è il robot Retriever per il committente?
Si tratta di un drr robot equipaggiato con appositi strumenti. *Che cosa fa il robot Retriever?*
Ha il compito di raggiungere la borsa individuata (contenente la bomba) e trasportarla alla posizione iniziale.

Entità	Interagisce con	Identificativo
Ddr robot discovery	Console	Discovery
Ddr robot Retriever	Console	Retriever
Console	Discovery, Retriever	Console

2.2.2 Analisi dei requisiti della fase di esplorazione

R-explore
Obbiettivo: Durante questa fase il robot deve esplorare la stanza, adducendo sufficienti informazioni per creare una rappresentazione del territorio comprendente ostacoli e posizioni degli ordigni. Emerge chiaramente fin da subito che l'output di questa fase consiste in una rappresentazione, che d'ora in poi chiameremo **Mappa**, che deve riportare le informazioni riguardanti la stanza e deve essere condivisa e compresa dagli altri componenti del sistema. In questa rappresentazione dovranno essere segnalate le posizioni di eventuali ostacoli, in modo che il robot Retriever possa ottimizzare il proprio percorso verso l'ordigno da rimuovere, la cui posizione indicata nella mappa deve essere, ovviamente, aggiornata in seguito alla rimozione dello stesso.

- *Requisito:* **R-startExplore**
  Entità coinvolte: Discovery, Console.
  Descrizione: l'esplorazione deve partire in seguito all'invio di un comando da parte dell'operatore.
- *Requisito:* **R-tempOk**
  Entità coinvolte: Discovery

Descrizione: l'esplorazione deve partire solo se la temperatura rispetta le condizioni prestabilite.

- *Requisito:* **R-stopExplore**
  Entità coinvolte: Discovery, Console.
  Descrizione: l'esplorazione deve stopparsi in caso l'utente lo richieda. Emerge quindi la necessità da parte del robot di essere reattivo ai comandi ricevuti.

- *Requisito:* **R-backHome**
  Entità coinvolte: Discovery, Console.
  Descrizione: il robot deve tornare alla posizione iniziale se richiesto dall'utente al seguito dello stop dell'esplorazione.

- *Requisito:* **R-continueExplore**
  Entità coinvolte: Discovery, Console.
  Descrizione: il robot deve continuare l'esplorazione se richiesto dall'utente.

- *Requisito:* **R-blinkLed**
  Entità coinvolte: Discovery.
  Descrizione: il led del robot deve lampeggiare se esso è in fase di esplorazione.

- *Requisito:* **R-consoleUpdate**
  Entità coinvolte: Discovery, Console.
  Descrizione: La console deve contenere le informazioni aggiornate riguardo allo stato di esplorazione del robot.

- *Requisito:* **R-stopAtBag**
  Entità coinvolte: Discovery.
  Descrizione: il robot deve fermarsi in prossimità di una valigia/ostacolo.

- *Requisito:* **R-takePhoto**
  Entità coinvolte: Discovery.
  Descrizione: il robot deve scattare una fotografia all'ostacolo appena individuato.

- *Requisito:* **R-sendPhoto**
  Entità coinvolte: Discovery, Console.

Descrizione: il robot deve inviare la fotografia appena scattata alla console.

- *Requisito:* **R-alert**
  Entità coinvolte: Console.
  Descrizione: il device dell'operatore, una volta ricevuta la foto, deve essere in grado di riconoscere se la valigia contiene una bomba, ed, in questo caso, inviare una notifica all'operatore.

- *Requisito:* **R-storePhoto**
  Entità coinvolte: Console.
  Descrizione: La fotografia deve essere salvata in modo permanente contestualmente alle informazioni relative.

- *Requisito:* **R-backHomeSinceBomb**
  Entità coinvolte: Discovery, Console.
  Descrizione: In caso la valigia contenga un ordigno ordinare al robot Discovery di tornare immediatamente alla posizione iniziale.

- *Requisito:* **R-continueExploreAfterPhoto**
  Entità coinvolte: Discovery, Console.
  Descrizione: se la valigia non contiene una bomba allora al robot viene comandato di continuare l'esplorazione, aggiornando contestualmente la Mappa del territorio.

2.2.3 Requisiti della fase di recupero

- *Requisito:* **R-waitForHome**
  Entità coinvolte: Discovery, Console.
  Descrizione: il sistema è in attesa che il robot discovery torni alla base.

- *Requisito:* **R-reachBag**
  Entità coinvolte: Retriever.
  Descrizione: il robot Retriever deve raggiungere la posizione dell'ordigno indicata nella mappa, ottimizzando il percorso al fine di evitare ostacoli già rilevati dal robot discovery. L'assunzione fatta è che il territorio sia rimasto invariato rispetto alla fase di esplorazione precedente.

- *Requisito:* **R-bagAtHome**
  Entità coinvolte: Retriever.

Descrizione: il robot Retriever dopo aver raccolto l'ordigno in un contenitore sicuro deve tornare alla base evitando gli ostacoli come in precedenza.

2.2.4 Output dell'analisi dei requisiti

Dopo aver elencato tutti i requisiti di base del sistema da realizzare, abbiamo deciso di definire delle priorità riguardo a quali requisiti analizzare più approfonditamente anche riguardo alle problematiche di progettazione. Criterio adottato per la scelta è quello di favorire i requisiti sia facili da realizzare che fondanti e di valore per il prodotto.
Abbiamo convenuto che al termine di questo sprint il committente sarebbe stato soddisfatto se gli avessimo fornito un sistema composto da console e robot explorer. L'esplorazione dovrebbe partire, se le condizioni ambientali lo consentono, dopo che l'utente tramite console ha inviato il comando appropriato. L'esplorazione in questo caso sarà d'esempio, soltanto per mostrare qualcosa all'utente, essa potrà essere interrotta in caso di invio del comando di stop e ripresa a piacimento dell'utente. I requisiti coinvolti, seppur estremamente semplificati sono i seguenti:

- **R-startExplore**: tramite una console grafica l'utente dovrà poter inviare un comando di inizio esplorazione, il comando dovrà essere ricevuto dal robot, il quale dovrà dare inizio alla fase di esplorazione se il requisito successivo è verificato.

- **R-Explore**: l'esplorazione per questo sprint iniziale consisterà in una serie di mosse dummy, in modo da permettere all'utente di vedere il robot muoversi.

- **R-tempOk**: l'esplorazione dovrà partire soltanto se la temperatura è inferiore ad una certa soglia. Da requisiti iniziali non è chiaro se il robot debba essere reattivo a cambi della temperatura durante la fase di esplorazione, inoltre non è specificato quale componente debba occuparsi sia del sensing che del checking della temperatura. Ci premureremo di chiarire questi aspetti con il committente.

- **R-stopExplore**: in seguito ad un comando dell'utente l'esplorazione dovrà essere interrotta ed il robot rimarrà in attesa di sapere dall'utente

se deve tornare a casa oppure proseguire l'esplorazione appena interrotta. Non è chiaro cosa debba fare il robot se non gli vengono fornite altre istruzioni.

- **R-continueExplore**: se l'utente ha inviato il comando per riprendere l'esplorazione il robot deve riprenderla.
- **R-backHome**: se invece l'utente ha deciso che il robot debba tornare alla sua base, il robot deve farlo. Specifichiamo però che in questo primo sprint non essendo stato ancora introdotto il concetto di mappa, il ritorno a casa sarà soltanto simulato, tuttavia il robot si troverà in uno stato in cui potrà iniziare una nuova esplorazione.

2.3 Analisi del problema

Partendo dai requisiti individuati e scelti in questo primo sprint, analizzeremo le problematiche relative a ciascuno di essi, individuando possibili approcci risolutivi analizzandone i pro e contro. Prima di analizzare i singoli requisiti dovremo però analizzare i componenti identificati in precedenza, tenendo conto di cosa ci fornisce il committente, cosa possiede la nostra software house e cosa dovremo realizzare. Una volta fatto questo dovremo affrontare la problematica dell'interazione fra componenti di un sistema distribuito eterogeneo.

2.3.1 Analisi della struttura del sistema

Console
Che cosa ci fornisce il committente?
Il committente lascia spazio alla nostra software house, esprimendo però, una preferenza rispetto ad una interfaccia che si renda disponibile da qualunque piattaforma. Alla luce delle indicazioni del committente siamo orientati ad utilizzare un'interfaccia web, così che possa essere fruita da qualsiasi dispositivo dotato di connessione ad internet.
Che cosa possiede la nostra software house?
La nostra software house possiede, da progetti precedenti, un server di frontend per applicazioni che coinvolgono un ddr-robot, utilizzando Node.js ed Express come tecnologia di riferimento. I vantaggi dell'utilizzo di questo applicazione web risiedono nel fatto che molti aspetti tecnologico-implementativi sono stati già realizzati e testati. Essendo basata su una rest api, non dovrebbe essere troppo complicato aggiungere funzionalità. Altro aspetto

positivo è il supporto nativo alla creazione di un modello del sistema, utilizzando lo standard JSON. Come contro saremmo vincolati alle specifiche scelte tecnologiche adottate, come ad esempio l'utilizzo di mqtt per comunicare con l'esterno e comunque servirà del tempo agli sviluppatori per familiarizzare con il prodotto già esistente, che potrebbe non esser stato mai visto da quegli specifici dipendenti.

Robot Discovery

Che cosa ci fornisce il committente?

Il committente ha già a disposizione un ddr robot fisico basato su un raspberry 3 dotato di wifi, con sitema operativo Raspbian(linux-based), capace di eseguire dei comandi basilari, quali andare avanti, indietro,a sinistra e a destra. Esso è inoltre dotato di un sonar posto anteriormente e di un led posteriore.

Che cosa possiede la nostra software house?

La nostra software house possiede delle librerie per comandare robot della tipologia fornitaci dal committente, più precisamente consiste in un layer software posto sopra pi4j in grado di tradurre comandi base (w,a,s,d,h) in azioni fisiche. Come pro abbiamo senz'altro un notevole risparmio di tempo e la possibilità di nascondere i dettagli implementativi della libreria pi4j, consentendo anche a chi non è esperto della materia di muovere correttamente il robot. Dato che probabilmente il robot explore coinciderà con il robot retriever per problemi di costi, potrebbero emergere situazioni problematiche legate all'utilizzo contemporaneo di pi4j da parte di due entità software.
La nostra software house possiede inoltre un ddr robot virtuale, legato al server di frontend citato precedentemente. Potrebbe essere conveniente sviluppare il software del sistema in modo che possa supportare entrambi i tipi di robot, così da poter utilizzare il robot virtuale al fine di testare la logica del sistema senza dover affrontare le problematiche tipiche dell'ambiente reale. Questa scelta potrebbe apportare notevoli benefici in termini di tempi di sviluppo e testing del sistema.

Robot Retriever

Che cosa ci fornisce il committente?

Il committente ci fornisce lo stesso ddr robot citato precedentemente, richiedendo che il sistema funzioni correttamente sia nel caso in cui sia un solo robot a doversi occupare di entrambe le mansioni (explore, retrieve), sia che in un secondo momento siano presenti due robot distinti.

2.3.2 Analisi dell'interazione nel sistema

Trattandosi di un sistema distribuito eterogeneo andranno analizzate nel dettaglio le interazioni tra i componenti del sistema, individuando le possibili strategie e valutandone la fattibilità, i costi, oltre che i vantaggi e gli svantaggi.
La comunicazione tra le varie entità costituisce un punto chiave nella realizzazione del sistema e al fine di promuovere la separazione dei compiti potrebbe essere utile introdurre nuove entità con lo scopo di gestire in modo adeguato le interazioni, o suddividere i componenti preesistenti in più sottoparti logicamente separate.
A questo punto è chiaro che emergono due sottosistemi separati all'interno del sistema: quello relativo alla console e quello relativo al robot, i quali dovranno comunicare tra di loro considerando però tutte le problematiche relative ai sistemi distribuiti.
Per quanto riguarda il sottosistema del robot può essere conveniente, dividere, fin da subito, il componente robot in due entità logiche distinte. Una delle due, che chiameremo **Explorer Mind** possiederà la business logic relativa ai compiti del Robot Explorer, mentre l'altra che defineremo **Player**, si occuperà di eseguire i comandi inviati dalla Explorer Mind. In questa fase emerge chiaramente che l'entità Player e Console si troveranno in due locazioni differenti, mentre per quanto riguarda l'Explorer Mind andrà individuato il miglior posizionamento possibile, valutando se posizionarlo insieme alla Console, insieme al Player o in una locazione terza.
Mantenendo l'Explorer Mind insieme alla Console, si ha un vantaggio nel caso in cui il Robot sia molto distante fisicamente e quindi modificarne il comportamento risulterebbe problematico. Si avrebbe inoltre a disposizione una rappresentazione virtuale del robot permettendone una più semplice consultazione. Per contro delegare tutta la business logic ad un componente esterno al robot, renderebbe scarsamente autonomo lo stesso nel caso di problemi di comunicazione. Inoltre la mole di messaggi scambiati via rete sarebbe molto maggiore, presentando anche problemi di latenze e possibili inconsistenze nei messaggi, tutto ciò potrebbe condurre il robot a non portare a termine il proprio compito. Al netto di queste considerazioni risulta preferibile la scelta di posizionare l'explorer mind insieme al player. Una volta individuate

ed analizzate le possibili strategie di posizionamento, occorre identificare la tipologia di comunicazione richiesta per ogni flusso di informazioni. Si hanno 2 canali di comunicazione, uno tra la Console e la Mind e uno tra la Mind e il Player. Per quanto riguarda la comunicazione a scambio di messaggi sarà opportuno definire una semantica appropriata, indicando informazioni utili come sorgente, destinatario, e distinguendo i messaggi a seconda della tipologia, che principalmente può essere dispatch (intendendo un messaggio point-to-point) ed evento (messaggio senza destinatario).

2.3.3 Analisi del comportamento del sistema

Una volta individuata la struttura e l'interazione dei componenti all'interno del sistema dovrà essere definito il comportamento del sistema che è l'insieme dei comportamenti e delle interazione delle singole entità. Al fine di ottenere una descrizione formale del comportamento del sistema verrà realizzata una macchina a stati finiti di Moore.

2.3.4 Analisi delle problematiche dei requisiti

In merito ai requisiti da soddisfare per queste fase, abbiamo fatto le seguenti considerazioni:

- **R-startExplore**: per soddisfare questo requisito sarà necessaria una comunicazione point to point, nel nostro gergo dispatch, tra Console e Mind.
- **R-Explore**: questo requisito coinvolge la Mind ed il Player, il quale dovrà attuare le azioni decise dalla prima. L'esplorazione è un problema complesso che ci riserviamo di trattare nel prossimo sprint.
- **R-tempOk**: analizzando questo requisito emerge la necessità di introdurre un nuovo componente atto a stabilire se le condizioni ambientali siano adeguate allo svolgimento dell'esplorazione. Dovendo osservare lo stesso ambiente in cui il robot agisce sarebbe preferibile posizionarlo nel sottosistema del robot.
- **R-stopExplore**: per soddisfare questo requisito sarà necessaria una comunicazione point to point tra Console e Mind.

- **R-continueExplore**: per soddisfare questo requisito sarà necessaria una comunicazione point to point tra Console e Mind. Una volta stoppata l'esplorazione il robot dovrà essere in grado di riprenderla in maniera consistente.
- **R-backHome**: essendo necessaria una rappresentazione del terreno non è ancora possibile realizzare a pieno questo requisito, che quindi verrà trattato nel successivo sprint.

2.4 Output dell'analisi del problema

Dopo aver individuato i 4 componenti logici del sistema, cioè **Robot Mind, Robot Actuator, Console e World Observer** e le interazioni di massima fra di essi, possiamo già realizzare un modello formale del sistema, tale per cui, pur senza scendere in dettagli implementativi, è già possibile dare un'adeguata visione del sistema. Questo schema potrà essere mantenuto durante tutto il processo, in quanto costituente uno scheletro di base su cui aggiungere dettagli in maniera monotona crescente. Per disambiguare la descrizione utilizzeremo il linguaggio **QA** per descrivere il modello del sistema. Il linguaggio QA si basa sul framework Akka, dove ogni QActor è un entità simile ad un attore Akka, con la caratteristica di essere message-based invece che message-driven, permettendo ad ogni attore di essere reattivo. Inoltre il linguaggio QA è adatto per costruire modelli eseguibili i sistemi software.
Verrà inoltre fornita una rappresentazione grafica del sistema con il solo scopo di facilitare la comprensione di struttura, interazione e comportamento dello stesso.

```
System robotWenvExplore
Dispatch startExploration : startExploration(X)
Dispatch haltExploration : haltExploration(X)
Dispatch resumeExploration : resumeExploration(X)
Dispatch backToHome : backToHome(X)
Dispatch envOk: envOk
Dispatch envNotOk: envNotOk
/*
 * EXPLORATION
 */
Dispatch doExplore     : doExplore
//Robot control messages
```

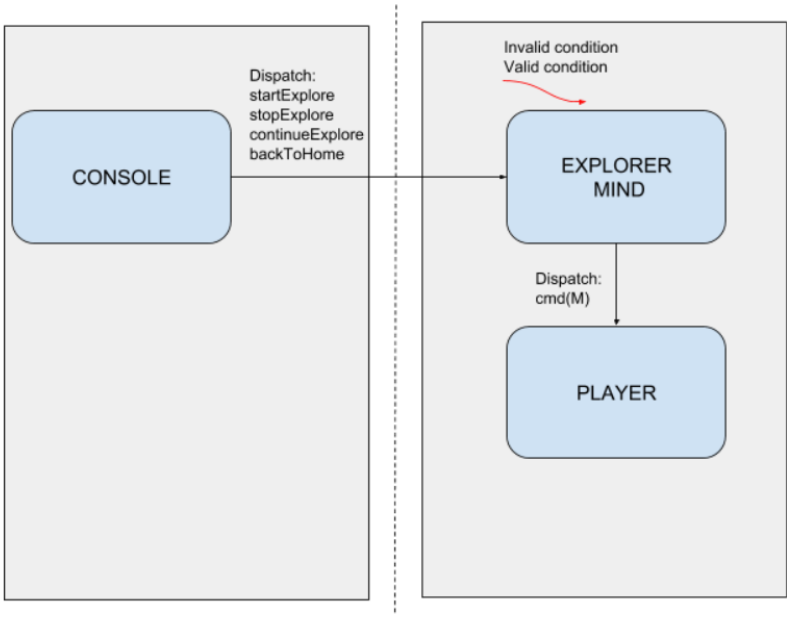

Figure 1: Schema sprint 1

```
Dispatch robotCmd       : robotCmd(M,T)
   //M=w|a|s|d|h  T:(int) duration

Event tempEvent : tempEvent(X)
Event sonarDetect : obstacle(X)      //From
   wenv-robotsonar-by clientTcpForVirtualRobot
Event sonar       : sonar(SONAR, TARGET, DISTANCE)
   //From wenv sonar
Event usercmd     : usercmd(X)     //from robot GUI;
   X=robotgui(CMD) CMD=s(low)
Dispatch userControl : cmd(X)     //from console


pubSubServer  "tcp://localhost:1883"

Context ctxRobotExplore      ip [ host="localhost"
   port=8028 ]

QActor console context ctxRobotExplore -pubsub{
```

```
    State init normal [  ]
    transition stopAfter 600000
    whenEvent usercmd -> handleUserCmd
    finally repeatPlan

    State handleUserCmd resumeLastPlan[
       printCurrentEvent;
            onEvent usercmd :
               usercmd(robotgui(w(SPEED))) -> forward
               mind -m robotCmd : robotCmd(w) ;
       onEvent usercmd : usercmd(robotgui(s(SPEED))) ->
          forward mind -m robotCmd : robotCmd(s) ;
       onEvent usercmd : usercmd(robotgui(a(SPEED))) ->
          forward mind -m robotCmd : robotCmd(a) ;
       onEvent usercmd : usercmd(robotgui(d(SPEED))) ->
          forward mind -m robotCmd : robotCmd(d) ;
       onEvent usercmd : usercmd(robotgui(h(SPEED))) ->
          forward mind -m robotCmd : robotCmd(h) ;

       onEvent usercmd : usercmd( explore ) -> forward
          mind -m startExploration :
          startExploration(ok);
       onEvent usercmd : usercmd( halt )    -> forward
          mind -m haltExploration  :
          haltExploration(ok);
       onEvent usercmd : usercmd( resume ) -> forward
          mind -m resumeExploration :
          resumeExploration(ok);
       onEvent usercmd : usercmd( backtohome )    ->
          forward mind -m backToHome  : backToHome(ok)
    ]
}
QActor worldobserver context ctxRobotExplore {
    Rules{
       temp(20).
    }
    Plan init normal[
       [?? temp(X)] emit tempEvent : tempEvent(X);
```

```
        delay 7000
    ]finally repeatPlan
}
QActor actuator context ctxRobotExplore {
    Plan init normal[
      javaRun
         it.unibo.robotVirtual.basicRobotExecutor.setUp("localhost");
      delay 1000;
      println("Actuator STARTS")
    ]
    switchTo  cmdIntepreter

    Plan cmdIntepreter[   ]
    transition stopAfter 600000
        whenMsg robotCmd -> execMove
    finally repeatPlan

    Plan execMove resumeLastPlan [
        println("Before the movement");
        onMsg robotCmd : robotCmd(M,T) -> javaRun
            it.unibo.robotVirtual.basicRobotExecutor.doMove(
            M,T );
        println("After the movement")
    ]
}


QActor mind context ctxRobotExplore  {
 Rules{
   temp(24).
   validCond( T,R ) :- temp(T), eval( lt, T, 26 ).
 }
 Plan init initial[
   println("Robot Mind Starts");
   [!? temp(X)] println(X)
 ]transition stopAfter 6000000
       whenEvent tempEvent : tempEvent(X) do
          ReplaceRule temp(_) with temp(X),
```

```
        whenMsg robotCmd : robotCmd(X) do forward
           actuator -m robotCmd : robotCmd(X),
        whenMsg startExploration -> checkEnvCondition
        finally repeatPlan

Plan checkEnvCondition[
  [ !? validCond(T,R)] {
     selfMsg envOk : envOk;
     println("Starting the exploration")
  } else selfMsg envNotOk: envNotOk
]transition whenTime 6000 -> init
whenMsg envOk -> exploration,
whenMsg envNotOk ->  init

Plan exploration resumeLastPlan[
  println("Before forward step");
  forward actuator -m robotCmd : robotCmd(w,2000);
  println("After forward step")
]transition stopAfter 600000
whenMsg haltExploration -> handleStop


Plan handleStop[
  println("Stopping the exploration, continue or back
     to home?");
  forward actuator -m robotCmd : robotCmd(h,10)
]transition whenTime 6000000 -> exploration
whenMsg resumeExploration -> exploration,
whenMsg backToHome -> backToHome

Plan backToHome[
  println("I'm coming back to home");
  forward actuator -m robotCmd: robotCmd(s,1000)
]switchTo init
}
```

2.5 Progettazione

In questa fase ci siamo occupati della progettazione delle soluzioni alle problematiche individuate nella fase precedente.

2.5.1 Progettazione della struttura del sistema

La prima questione relativamente alla struttura del sistema riguardava la scelta dell'utilizzo del server frontend. Dopo averne compreso la compatibilità con il problema da risolvere è stato deciso di utilizzare questa applicazione.
Uno dei motivi che ha spinto al riutilizzo del frontend è il fatto che si basa sul pattern architetturale MVC, permettendo di intervenire più facilmente sui singoli moduli, permettendo di cambiarne il comportamento senza dover effettuare modifiche onerose. La scelta di questo modulo comporta inoltre l'utilizzo della tecnologia MQTT, vincolo che non ha comportato eccessivi problemi di progettazione per il fatto che i QA possiedono nativamente un supporto per la comunicazione publish/subscribe.
Al fine di velocizzare i tempi di realizzazione del sistema, evitando di dover affrontare problematiche relative al robot fisico, è stato utilizzato, per questo primo sprint, il robot virtuale a dispoizione, realizzando comunque il sistema in modo che l'utilizzo del robot fisico potesse essere effettuato con sforzo esiguo.

2.5.2 Progettazione dell'interazione nel sistema

Tenendo conto del vincolo derivante dall'utilizzo di MQTT è stato adottato come principio realizzativo quello di utilizzare una comunicazione di tipo publish/subscribe per la comunicazione tra diversi contesti, mentre per la comunicazione intra-contesto effettuare un normale scambio di messaggi.
Al fine di realizzare un consistente protocollo applicativo è stata individuata una precisa semantica dei messaggi, dove ogni messaggio è della forma:

msg(MSGID, MSGTYPE, SENDER, RECEIVER, CONTENT, SEQNUM)

dove:

- **MSGID**: identificatore del messaggio
- **MSGTYPE**: tipo di messaggio (dispatch, event, request)

- **SENDER**: identificatore del mittente
- **RECEIVER**: identificatore del destinatario
- **CONTENT**: contenuto del messaggio
- **SEQNUM**: numero univoco associato al messaggio

Una importante nota va aggiunta per la differenza di utilizzo tra il tipo del messaggio. Sono stati utilizzati messaggi di tipo dispatch ogni volta che il messaggio fosse indirizzato ad un destinatario ben preciso, mentre i messaggi di tipo event nel caso contrario.
Un altro problema di cui andava effettuata la scelta era relativamente al posizionamento della Explorer Mind, la quale è stata collocata insieme al Player per i vantaggi analizzati nella fase precedente. Perciò l'interazione risultante è formata da due canali:

- **Console - Explorer Mind**: questo canale di comunicazione dovrà permettere l'interazione tra due entità di due tecnsologie diverse in locazioni separate, perciò attraverso lo scambio di messaggi e mediante MQTT.
- **Explorer Mind - Player**: in questo caso poterebbe non essere necessario ricorrere allo scambio di messaggi, tuttavia, per promuovere l'estendibilità del sistema, potrebbe essere preferibile utilizzare questo approccio, in modo da poter spostare il collocamento della Mind senza ripercussioni sul sistema.

Relativamente al contesto del robot è stato deciso di introdurre un'ulteriore entità, di secondaria importanza, da collocare al bordo del sottosistema. Tale entità è adibita alla ricezione dei comandi dell'utente, che vengono poi convertiti e rigirati alla Mind.

2.5.3 Progettazione del comportamento del sistema

Come anticipato in precedenza il comportamento del sistema può essere descritto mediante automi a stati finiti di Moore, a differenti livelli di astrazione. I QA inoltre possiedono un supporto nativo per la realizzazione e consultazione di una base di conoscenza in prolog. Su questa base di conoscenza possono essere effettuate tutte le operazioni principali, come l'aggiunta di

regole o l'esecuzione di query.
Tutte le informazioni utili ai QActor che devono essere mantenute nel tempo verranno memorizzate in questa modalità.

2.6 Sprint Review

In questa fase abbiamo interagito con il committente al fine di chiarire i dubbi emersi:

- In merito al requisito R-tempOk non era chiaro se il robot dovesse essere reattivo al cambiamento delle condizioni ambientali durante l'esplorazione. Il committente ha espresso la preferenza che fosse reattivo al cambio di temperatura.

- In merito alla reattività alle condizioni ambientali il committente ci ha chiarito che il robot dovrebbe tornare a casa se le condizioni non sono più valide. Esso però si riserva di cambiare successivamente la specifica e richiede quindi che un eventuale cambiamento sia facilmente attuabile.

- In merito al requisito R-stopExplore il committente ha richiesto di far proseguire l'esplorazione se dopo un tempo molto elevato non vengono ricevute istruzioni. Esso però si riserva di cambiare successivamente la specifica e richiede quindi che un eventuale cambiamento sia facilmente attuabile.

3 Sprint 2

3.1 Sprint Planning

Alla fine di questo sprint vorremmo ottenere un sistema che realizzi tutte le funzionalità principali legate ai compiti del Robot Explore, tralasciando soltanto quelle più implementative e meno concettuali. Andranno quindi analizzati con precisione i requisiti e le relative problematiche non trattati nello sprint precedente, per poi progettare le parti del sistema coinvolte. Output di questo sprint dovrà essere un sistema che compie correttamente tutta la fase di esplorazione dell'ambiente ed individuazione delle bombe, unita ad una comunicazione puntuale dello stato del robot alla console. Come

da buona pratica dovremo inoltre predisporre un piano di testing da attuare, considerando però la problematica del testing distribuito.

3.2 Analisi dei requisiti

Come già introdotto nello sprint planning in questo sprint vogliamo analizzare i requisiti che ci permettano di mostrare al committente un sistema in cui l'esplorazione parte dopo che l'utente ha inviato l'apposito comando e solo se le condizioni ambientali sono verificate. Dopo il chiarimento fornitoci il robot dovrà essere reattivo al mutare delle condizioni ambientali, tornando quindi a casa in caso di condizioni avverse. Al seguito dell'invio del comando di stop il robot dovrà fermarsi e, se richiesto dall'utente, tornare alla sua base. Se durante l'esplorazione il robot incontrerà degli ostacoli dovrà farne segnalazione alla console, mettendosi in attesa di ulteriori istruzioni, che gli indicheranno se continuare l'esplorazione perchè l'ostacolo incontrato non è una bomba oppure di tornare alla sua base in quanto l'ostacolo è una bomba. Di seguito i requisiti coinvolti:

- *Requisito:* **R-tempOk**
  Entità coinvolte: Discovery
  Descrizione: l'esplorazione deve partire solo se la temperatura rispetta le condizioni prestabilite. Inoltre l'esplorazione deve interrompersi se le condizioni non sono più valide.

- *Requisito:* **R-explore**
  Entità coinvolte: Discovery
  Descrizione: l'esplorazione dovrà proseguire per passi, premurandosi di costruire una rappresentazione fedele dell'ambiente, evitando gli ostacoli individuati in precedenza e coprendo progressivamente tutto l'ambiente.

- *Requisito:* **R-stopExplore**
  Entità coinvolte: Discovery, Console.
  Descrizione: l'esplorazione deve stopparsi in caso l'utente lo richieda. Emerge quindi la necessità da parte del robot di essere reattivo ai comandi ricevuti. Il robot dovrà quindi mettersi in attesa di ulteriori istruzioni che gli indicano se continuare l'esplorazione oppure tornare alla base. Se dopo un certo tempo queste indicazioni non arrivassero allora il robot dovrà continuare l'esplorazione.

- *Requisito:* **R-backHome**
  Entità coinvolte: Discovery, Console.
  Descrizione: il robot deve tornare alla posizione iniziale se richiesto dall'utente al seguito dello stop dell'esplorazione. Dovrà quindi essere consapevole della sua attuale posizione ed elaborare un percorso per tornare a quella che è contrassegnata come la sua base.

- *Requisito:* **R-continueExplore**
  Entità coinvolte: Discovery, Console.
  Descrizione: il robot deve continuare l'esplorazione se richiesto dall'utente. Dovrà quindi proseguire ad attuare le strategie di esplorazione che aveva scelto di adottare prima di essere fermato.

- *Requisito:* **R-blinkLed**
  Entità coinvolte: Discovery.
  Descrizione: il led del robot deve lampeggiare se esso è in fase di esplorazione. Il committente ci ha chiarito che il led sarà situato sul robot stesso, ma in un secondo momento potrebbe essere anche un led esterno al robot.

- *Requisito:* **R-consoleUpdate**
  Entità coinvolte: Discovery, Console.
  Descrizione: La console deve contenere le informazioni aggiornate riguardo allo stato del robot. Le informazioni da visualizzare all'utente saranno, per adesso, se il robot sta esplorando oppure no e la temperatura dell'ambiente.

- *Requisito:* **R-stopAtBag**
  Entità coinvolte: Discovery.
  Descrizione: il robot deve fermarsi in prossimità di una valigia/ostacolo ed informare la console dell'accadimento, aspettando di sapere da essa se l'ostacolo è una bomba e, perciò, tornare a casa, oppure continuare l'esplorazione appuntandosi che quella specifica posizione contiene un ostacolo.

- *Requisito:* **R-alert**
  Entità coinvolte: Console.
  Descrizione: il device dell'operatore, una volta ricevuta la foto, deve essere in grado di riconoscere se la valigia contiene una bomba, ed, in

questo caso, notificarlo al robot. L'ultima scelta riguardo alla classificazione dell'ostacolo sarà però in carico all'utente.

- *Requisito:* **R-backHomeSinceBomb**
  Entità coinvolte: Discovery, Console.
  Descrizione: In caso la valigia contenga un ordigno ordinare al robot Discovery di tornare immediatamente alla posizione iniziale.

- *Requisito:* **R-continueExploreAfterPhoto**
  Entità coinvolte: Discovery, Console.
  Descrizione: se la valigia non contiene una bomba allora al robot viene comandato di continuare l'esplorazione, aggiornando contestualmente la Mappa del territorio.

3.3 Analisi del problema

In merito ai requisiti da soddisfare per queste fase, abbiamo fatto le seguenti considerazioni:

- **R-tempOk**: al fine di rendere il robot reattivo al cambiamento delle condizioni ambientali occorre effettuare un'estensione di quanto realizzato nello sprint precedente, individuando le possibili strategie per far sì che il robot reagisca correttamente. Emerge in maniera evidente che la variazione delle condizioni ambientali deve essere trattata come un evento, infatti dallo sprint precedente il componente adibito al controllo delle condizioni (World Observer) deve rendere disponibili le informazioni iniettandole nell'ambiente senza premurarsi di conoscere gli interessati.
  Il robot, pur essendo impegnato nell'esplorazione, deve poter reagire prontamente alla variazione di temperatura. Una soluzione potrebbe essere quella di chiedere prima di ogni passo esplorativo la validità delle condizioni, mentre come altro approccio si potrebbe introdurre un meccanismo che permetta la registrazione in ascolto di tali eventi. In questo caso, la ricezione di tali eventi produrrà una variazione sulla conoscenza del mondo da parte del robot.

- **R-explore**: le problematiche principali relative a questo requisito riguardano la costruzione di una rappresentazione fedele del mondo, che chiameremo mappa e la scelta del criterio di esplorazione. Per soddisfare questo

requisito sono possibili diverse alternative, una delle quali potrebbe essere l'utilizzo di sensoristica adibita al tracciamento del robot indoor. Un'alternativa consiste nello sviluppo di un sistema totalmente software che si basa sui movimenti effettuati da robot.
Che cosa possiede la software house? La software house possiede un modulo software per la costruzione di una mappa basandosi sui movimenti effettuati dal robot. Tali movimenti vengono determinati da un'intelligenza artificiale, in grado di fornire le mosse da eseguire per raggiungere un determinato obiettivo.
La mappa in questione è costituita da celle della dimensione del robot, ognuna contrassegnata da un simbolo che indica la tipologia del contenuto, il quale può essere cella libera, cella da esplorare, cella con ostacolo o cella con robot. Considerato l'equipaggiamento del robot fornitoci, risulta quasi obbligatorio scegliere una via totalmente software, valutando se sia possibile integrare il modulo in possesso della nostra software house. Lo sviluppo di un modulo simile da zero richiederebbe notevoli sforzi che potrebbero rallentare notevolmente il processo di sviluppo. Il robot dovrà essere a conoscenza della sua posizione attuale rispetto alla mappa che sta creando.
Dovrà essere predisposta una utility che permetterà di ottenere le azioni da compiere per raggiungere una determinata cella della mappa.
Sebbene non sia l'obbiettivo di questo requisito, andrà posta particolare attenzione alla modellazione della mappa, tenendo conto che essa sarà d'interesse anche per altre parti del sistema.

- **R-backHome**: le problematiche legate a questo requisito sono del tutto simili a quelle relative al R-explore, ovvero è necessario individuare le strategie di costruzione della mappa e le interazioni con il modulo dedicato al fornimento delle azioni da compiere per raggiungere un obiettivo, che in questo caso sarà la base del robot. Una volta giunto a casa dovrà permettere di iniziare una nuova esplorazione tenendo conto delle informazioni precedentemente ottenute.
  Dopo aver analizzato il modulo di planning abbiamo scoperto che le azioni forniteci hanno l'obbiettivo di esplorare l'ambiente anche durante la fase di ritorno a casa. Questo può provocare situazioni problematiche nel caso in cui si incontri un ostacolo nella strada del ritorno. In questo caso andrà studiato il corretto modo di reagire a questa situazione, eventualmente anche chiedendo chiarimenti al com-

mittente. Una possibile alternativa è quella di estendere il modulo di planning introducendo una strategia più conservativa che permetta di raggiungere la base utilizzando solamente celle già note.

- **R-blinkLed**: nel caso in questione il committente fornisce un led che si trova a bordo del robot, tuttavia in un secondo momento la posizione del led potrebbe variare, per cui andrà introdotto un nuovo componente logico demandato alla gestione del led. La logica secondo la quale il led lampeggia o è spento deve essere modellata in base allo stato del robot, motivo per il quale, già in questa fase riteniamo opportuno separarla dalla gestione fisica del led. Questa logica potrebbe essere inserita o nella Explorer Mind oppure nel World Observer.

- **R-consoleUpdate**: per quanto riguarda l'aggiornamento della console andrà studiata la modalità di interazione fra Robot e Console. Considerato che le informazioni da visualizzare all'utente sono la temperatura dell'ambiente e lo stato del Robot una possibile strategia consiste nell'affidare al World Observer il compito di inviare tali dati, in quanto già possiede le informazioni relative alla temperatura e potrebbe ricevere dal robot il suo stato. Questa soluzione presenta il vantaggio di scaricare la Mind dal compito di aggiornare l'utente con informazioni che non lo riguardano direttamente (temperatura). Pur non essendo tra i requisiti di base, potrebbe essere di valore per l'utente mostrare in diretta la mappa costruita dal robot. Andrà perciò valutato se sia utile realizzare modelli diversi della mappa a seconda del contesto coinvolto o se sia sufficiente mostrarne solo una rappresentazione grafica. Nella nostra valutazione dovremo tener conto del modello già esistente nel modulo della nostra software house relativo alla mappa in quanto potrebbe non essere adatto a tutte le nostre necessità.

- **R-stopAtBag**: considerando che l'individuazione degli ostacoli da parte del robot deve avvenire mediante un sensore di prossimità (che può essere fisico o virtuale), occorrerà reagire correttamente alle informazioni da esso emesse. Occorrerà inoltre verificare la compatibilità del software in possesso della software house con la tipologia di robot che adotteremo.
  Andranno inoltre gestiti i casi in cui durante l'esplorazione il robot non riesca a completare la sua mossa a causa di un ostacolo rilevato, queste

situazioni, se gestite in modo superficiale, potrebbero dar luogo ad una rappresentazione dell'ambiente incongruente.

- **R-backHomeSinceBomb**: anche in questo caso valgono le considerazioni fatte per il requisito R-backHome, tuttavia andrà effettuata una differenziazione tenendo conto che in questo caso il robot una volta raggiunta la base dovrà fermarsi e restare in attesa che il Robot Retriever recuperi l'ordigno.

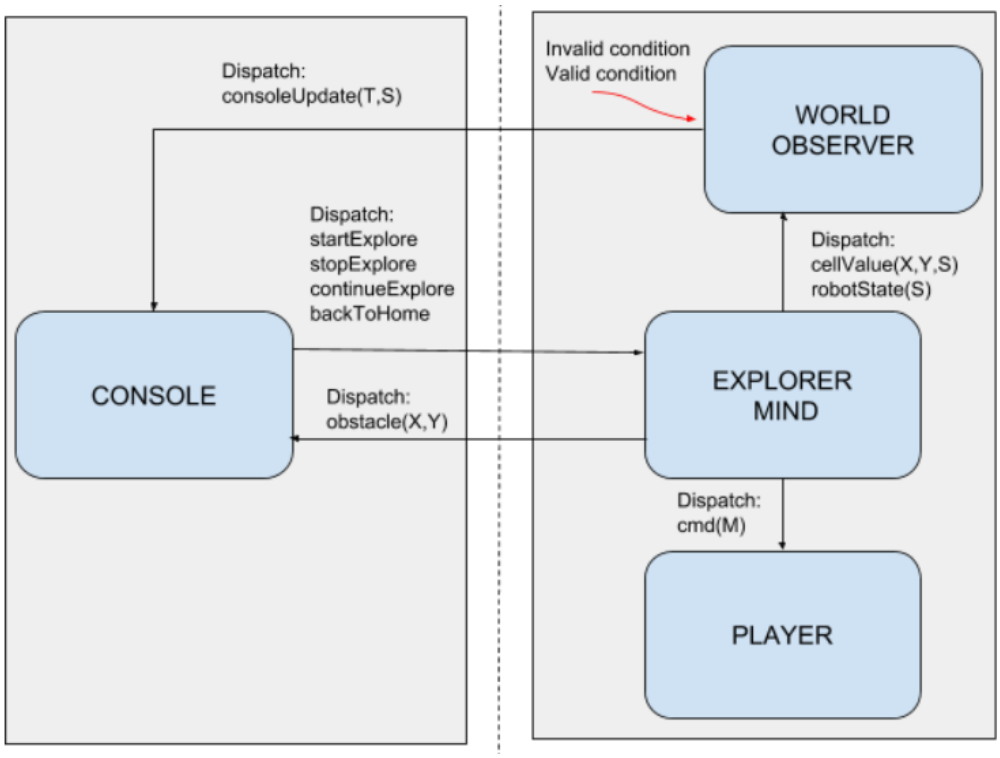

Figure 2: Schema sprint 2

3.4 Test Planning

La problematica del testing nei sistemi distribuiti eterogenei è sempre argomento di discussione e rappresenta uno scoglio abbastanza importante. Nel nostro caso infatti è stato possibile realizzare dei test automatizzati solo per

alcuni dei requisiti da soddisfare, cioè quelli che non coinvolgono la valutazione dell'interazione con il mondo fisico. Per svolgere i test si è rivelata molto utile il supporto fornitoci dal modulo **QActorUtils**, che ci ha garantito la possibilità di ispezionare la base di conoscenza interna, mandare messaggi, ottenere i riferimenti, ai QActors di un determinato contesto. E' evidente come i test automatici siano parziali e non esaustivi, ma possono essere già di aiuto in quelle situazioni in cui vogliamo testare l'interazione ed il comportamento complessivo del sistema. Per la natura dei requisiti è stato possibile automatizzare i test per i requisiti:

- **TestRBackHome**: si vuole accertare che dopo aver fatto partire l'esplorazione, atteso un paio di secondi e stoppata la stessa, se viene ricevuto il comando per tornare alla base il robot dopo un certo lasso di tempo si trova esattamente a casa.
- **TestRBlinkLed**: si vuole accertare che se il robot è in fase di esplorazione il led lmapeggi, mentre se è fermo allora il led sia spento.
- **TestRContinueExploration**: si vuole accertare che dopo aver fatto partire l'esplorazione, atteso un paio di secondi e stoppata la stessa, se viene ricevuto il comando per continuare l'esplorazione lo stato del robot deve essere quello di esplorazione.
- **TestROkTemperature:** viene testato che se la temperatura è superiore alla soglia, anche se il robot explorer riceve il comando per iniziare l'esplorazione, il suo stato non cambi.
- **TestRStartExplore:** viene testato che quando il robot explorer riceve il comando per partire effettivamente il suo stato diventi di esplorazione.
- **TestRStopExplore:** si vuole accertare che dopo aver fatto partire l'esplorazione, atteso un paio di secondi e stoppata la stessa, lo stato di robot non sia quello di esplorazione.

Per quanto riguarda gli altri requisiti, che includono l'interazione con l'ambiente, gli unici test che possiamo condurre sono di tipo qualitativo. Ciò non toglie che vanno comunque considerati tutti i possibili casi di utilizzo, in modo da fornire all'utente un prodotto adeguato. Andrà quindi verificato che:

- La mappa creata sia coerente con la realtà.

- Una volta individuato un ostacolo il robot nelle sue mosse successive non deve più imbattercisi.
- Il robot torni effettivamente nella sua base dopo che gli è stato richiesto.
- Il robot sia a casa dopo un certo lasso di tempo che le condizioni sono invalide e non è quindi possibile proseguire l'esplorazione.
- Il robot dia adeguata comunicazione alla console se incontra un ostacolo.
- Il robot stia effettivamente compiendo le azioni previste dal suo stato.

3.5 Progettazione

- **R-tempOk**: per quanto riguarda il controllo delle condizioni ambientali è stato deciso di modificare il World Observer affidandone tale compito. A fronte di un cambiamento di temperatura, infatti, il World Observer verifica le condizioni ed emette un evento che indica la validità delle stesse. Un'entità apposita poi si occupa di ascoltare tale evento andando a modificare la conoscenza del mondo della Mind modificandone la base di conoscenza. Ad ogni passo esplorativo perciò la Mind consulta la base di conoscenza per capire se può continuare l'esplorazione oppure no.
- **R-explore**: dopo aver considerato e studiato le varie alternative emerse in fase di analisi del problema, abbiamo ritenuto preferibile adottare un approccio totalmente software, basato sul modulo già esistente posseduto dalla software house. Questo modulo fa uso di una libreria esterna denominata AIMA, che fornisce un AI di base per l'ottenimento delle azioni da svolgere per raggiungere un goal. Nel modulo è già presente un modello di mappa interamente realizzato in Java, utilizzato per individuare i percorsi da intraprendere durante l'esplorazione. Abbiamo deciso che per via dei futuri requisiti può essere utile costruire una rappresentazione della mappa in prolog, da inserire nella base di conoscenza del World Observer, che a questo punto avrà anche conoscenza della mappa dell'ambiente. Per utilizzare il modulo di planning verrà comoda la capacità nativa dei QA di eseguire codice java. Quindi dato un goal il modulo inserirà nella base di conoscenza del richiedente le mosse da eseguire per soddisfare quell'obbiettivo.

- **R-backHome**: Per tornare a casa dopo un halt inviato dall'utente abbiamo utilizzato il modulo citato in precedenza, ponendo come goal quello di ritornare alla posizione iniziale (0,0). Abbiamo deciso che per il momento utilizzeremo il planner così com'è, senza apportare modifiche, ma sarà nostra cura nel prossimo sprint trovare una soluzione per utilizzare un approccio conservativo nello stabilire il percorso di ritorno.

- **R-blinkLed**: per quanto riguarda la gestione dello stato del led è stato deciso di affidarne la logica al World Observer. I vantaggi di questa soluzione risiedono principalmente nel fatto che se si decidesse di spostare fisicamente il led in un altra locazione non deve essere la mind ad occuparsi della gestione della logica di un componente ad essa esterno.

- **R-consoleUpdate**: abbiamo deciso di affidare al World Observer il compito di inviare i dati alla console, in quanto è a conoscenza di tutte le informazioni necessarie per presentare la situazione dell'ambiente (incluso il robot) all'utente. Come vantaggio di questo approccio avremo un punto chiaro e definito dove ricercare tutte le informazioni relative al sistema, minimizzando al contempo gli entry point del sistema.
  Per quanto riguarda la visualizzazione della mappa nel sottosistema della console abbiamo deciso di realizzare un modello di essa, rispettando il pattern MVC proposto dal modulo. In questo modo al variare del modello si avrà una variazione della rappresentazione grafica.

- **R-stopAtBag**: in questo caso occorre affrontare il problema relativo al caso in cui il robot incontri ostacoli prima di aver completato la cella. Abbiamo deciso che il robot, per evitare incongruenze, deve tornare indietro fino alla cella precedente e segnalare quindi la cella come ostacolo. Tale scelta è anche motivata dal fatto che la nostra software house possiede già un modulo software che realizza questa feature.

- **R-backHomeSinceBomb**: Per quanto riguarda il ritorno a casa del robot in seguito al rilevamento di una bomba abbiamo realizzato un differente piano per il ritorno a casa in modo da assicurare che non si possa iniziare una nuova esplorazione al suo termine. Anche in questo

caso andrà valutata la realizzazione di un approccio conservativo per il ritorno.

3.6 Sprint Review

In seguito al confronto con il committente sono emerse alcune preferenze. Il committente in primo luogo si premura che venga realizzata la feature relativa all'invio della foto scattata alla console. Inoltre viene chiesto che il robot dopo aver incontrato una bomba intraprenda un percorso di ritorno considerato sicuro, cioè privo di ostacoli o celle non esplorate. Infine richiede un'interfaccia grafica più visivamente gradevole e semplice, in cui la mappa si dimensioni dinamicamente coerentemente con l'esplorazione.

4 Sprint 3

4.1 Sprint Planning

Alla fine di questo sprint vorremmo ottenere un sistema che realizzi tutte le funzionalità richieste dal committente. Output di questo sprint dovrà essere un sistema in cui sia presente anche il Robot Retriever in grado di effettuare le azioni previste.

4.2 Analisi dei requisiti

- *Requisito:* **R-takePhoto**
  Entità coinvolte: Discovery.
  Descrizione: il robot deve scattare una fotografia all'ostacolo appena individuato.

- *Requisito:* **R-sendPhoto**
  Entità coinvolte: Discovery, Console.
  Descrizione: il robot deve inviare la fotografia appena scattata alla console.

- *Requisito:* **R-storePhoto**
  Entità coinvolte: Console.
  Descrizione: Se la fotografia ricevuta dalla console rappresenta un ostacolo classificato come bomba deve essere salvata in modo permanente

contestualmente alle informazioni relative. Le informazione in questione sono la posizione della bomba e il tempo alla rilevazione della bomba. Non viene specificato quale debba essere la modalità e la posizione di salvataggio. Il committente ci fornisce però libertà su questo.

- *Requisito:* **R-waitForHome**
  Entità coinvolte: Discovery, Console.
  Descrizione: il sistema è in attesa che il robot discovery torni alla base, trovandosi in una condizione che non consenta di effettuare alcuna azione.

- *Requisito:* **R-reachBag**
  Entità coinvolte: Retriever.
  Descrizione: il robot Retriever deve raggiungere la posizione dell'ordigno indicata nella mappa, ottimizzando il percorso al fine di evitare ostacoli già rilevati dal robot discovery. L'assunzione fatta è che il territorio sia rimasto invariato rispetto alla fase di esplorazione precedente.

- *Requisito:* **R-bagAtHome**
  Entità coinvolte: Retriever.
  Descrizione: il robot Retriever dopo aver raccolto l'ordigno in un contenitore sicuro deve tornare alla base evitando gli ostacoli come in precedenza.

4.3 Analisi del Problema

- **R-sendPhoto**: per questo requisito le problematiche principali riguardano la modalità di conversione dell'immagine, occorre perciò identificare le possibili strategie per ottenere una rappresentazione adeguata alla tipologia di comunicazione adottata.

- **R-waitForHome**: questo requisito rappresenta un punto critico per quanto riguarda l'interazione tra i componenti, essendo infatti tutti coinvolti. Il robot explorer dovrà dare comunicazione del suo avvenuto ritorno alla base, a questo punto il robot retriever dovrà essere messo nelle condizioni di eseguire il suo compito. E' quindi evidente che debba essere risolto il problema della comunicazione della mappa dedotta nella fase di esplorazione al robot retriever che non conosce l'ambiente dove dovrà operare. Dovrà inoltre essere trasferita la conoscenza della

posizione dell'ordigno da raggiungere. Andranno valutate le possibili strategie di scambio del modello della mappa.

- **R-reachBag**: in questo caso è fondamentale che il Robot Retriever sia in grado di raggiungere l'ordigno evitando ostacoli e celle non esplorate, per questo motivo diventa sempre più stringente l'estensione del planner con un approccio più conservativo.
- **R-bagAtHome**: anche in questo caso è richiesto che il robot raggiunga la base tramite un percorso sicuro. Valgono perciò le considerazioni effettuate per il requisito precedente.

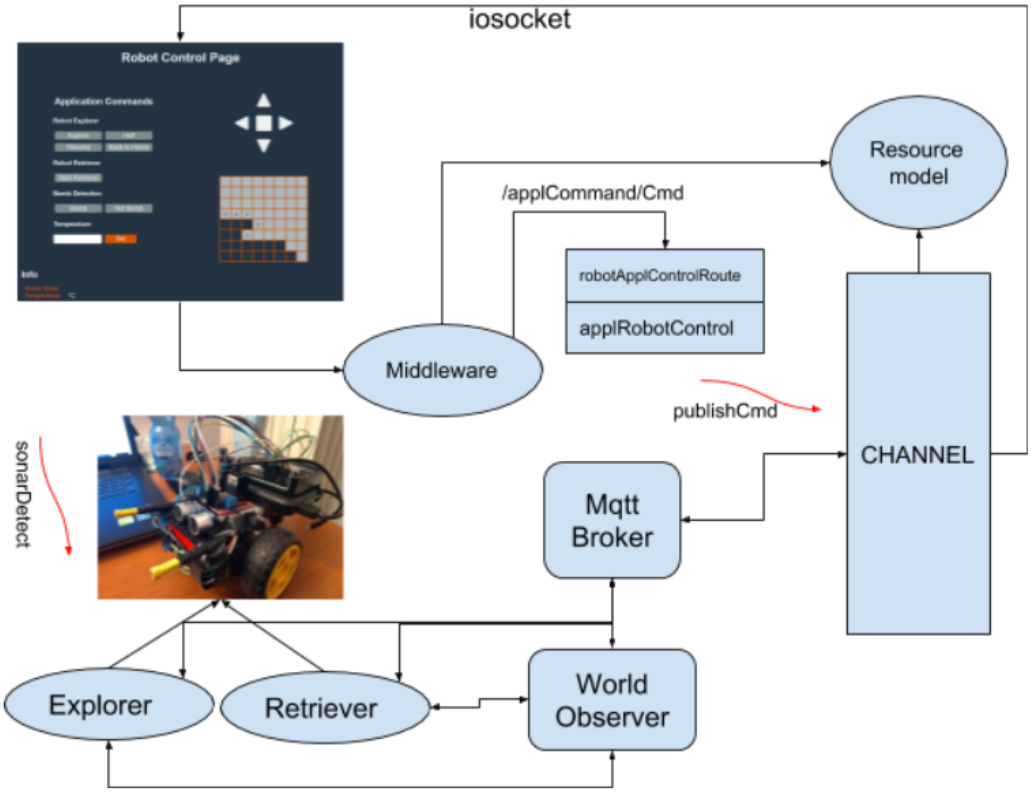

Figure 3: Schema sprint 3

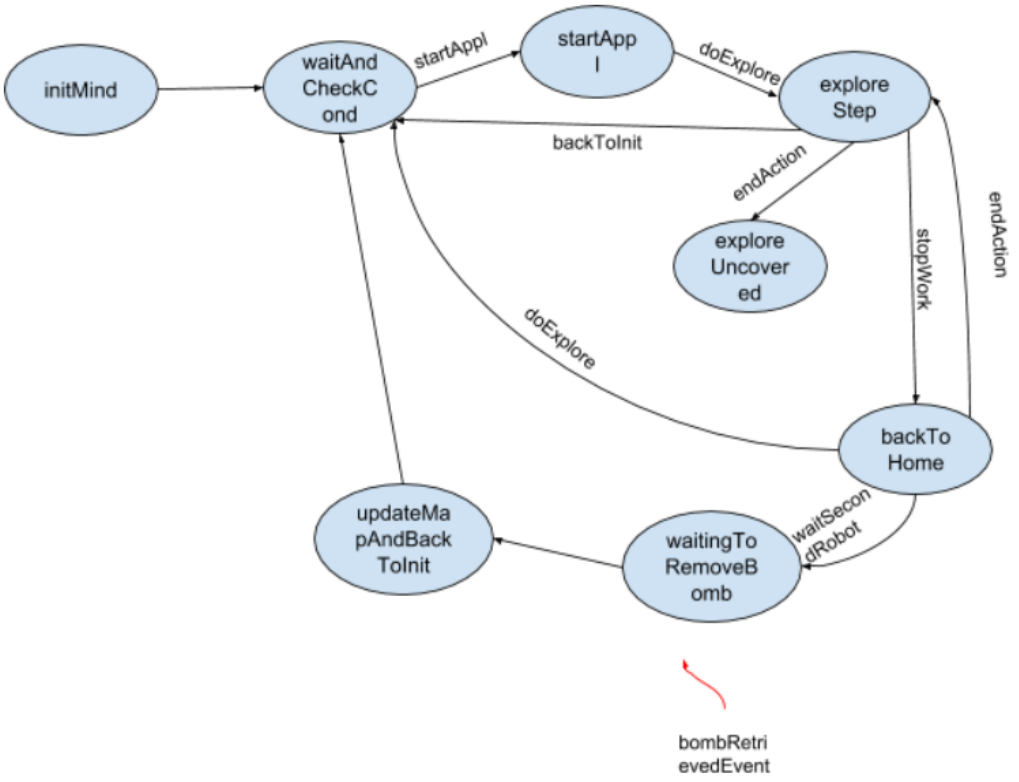

Figure 4: Automa a stati finiti Mind sprint 3

4.4 Progettazione

- **R-sendPhoto**: per la realizzazione di questo requisito si è deciso di convertire la stringa in formato base64, in modo da permettere l'invio della foto sotto forma di stringa secondo formato standard.

- **R-waitForHome**: lo scambio di informazioni avviene in modo del tutto simile a quanto già effettuato in precedenza, totalmente a scambio di messaggi e favorendo l'utilizzo di MQTT per contesti differenti. Per quanto riguarda l'invio della rappresentazione della mappa è stato deciso di effettuare lo scambio da parte del Robot Explorer, il quale invia un messaggio al Robot Retriever contenente tutte le informazioni relative a tutte le celle della mappa. Il Retriever a questo punto inietta tali informazioni nella propria base di conoscenza e tramite Javarun

richiama apposite funzioni per costruire il modello della mappa che dovrà essere utilizzato dall'AI. A questo punto il retriever è pronto per compiere la propria missione di recupero della bomba.

- **R-reachBag**: nella realizzazione di tale requisito abbiamo apportato alcune modifiche al planner della software house in modo da poter introdurre una nuova strategia più conservativa. Tali modifiche consistono essenzialmente nell'introduzione di una nuova classe Java del tutto simile a quella precedentemente utilizzata per la valutazione della possibilità dell'esplorazione di una data cella. Se infatti nell'implementazione precedente veniva permessa l'esplorazione di una cella non ancora visitata durante la fase di ritorno, in questo caso viene negata.

- **R-bagAtHome**: la modifica realizzata per il requisito precedente è stata riutilizzata per questo caso come per altri precedentemente citati.

4.4.1 Progettazione del robot fisico

Relativamente al robot fisico occorreva effettuare una scelta per ciò che riguardava la conversione dei comandi in azioni fisiche. La scelta riguardava l'utilizzo di una libreria a disposizione oppure la realizzazione di un modulo custom.
A seguito di un attento studio è emerso che esistevano alcuni i problemi derivanti dall'utilizzo dello stesso robot per le due entità logicamente distinte, più precisamente riguardo all'inizializzazione delle configurazioni.
Considerando inoltre che il robot deve compiere solamente un piccolo insieme di movimenti facilmente configurabili è stato deciso di non utilizzare la libreria già a disposizione creando alcuni script adibiti all'attuazione del movimento indicato.