Ingegneria dei Sistemi Software
A differential drive Robot
Seconda Parte

Beatrice Mezzapesa, Alessia Papini, Lorenzo Pontellini

Alma Mater Studiorum – University of Bologna
via Venezia 52, 47023 Cesena, Italy
{beatrice.mezzapesa, alessia.papini, lorenzo.pontellini}@studio.unibo.it

1 Introduction

Attraverso l'uso del seguente report, si vogliono esprimere i fatti e le interazioni avvenute nella gestione e sviluppo di un sistema software per il controllo di un robot in ambiente protetto. Un ulteriore scopo è quello di fornire uno storico per la gestione del processo produttivo del sistema software esprimendo fatti rilevanti attraverso l'uso di modelli formali interpretabili anche da personale non tecnico. Ci si avvale inoltre del supporto di un meta modello custom che permette di realizzare prototipi funzionanti abbattendo i tempi di testing del sistema. Il compimento e la gestione del seguente progetto si portano dunque al quarto livello dello standard CMM (Capability Maturity Model) cioè processo produttivo managed. Questo sta a significare come l'organizzazione sia capace di costruire prodotti software, impostando una fase di predizione dei costi e del piano di lavoro, basandosi su una classificazione dei compiti e dei componenti e su metriche di misura dei loro costi e tempi di sviluppo.

2 Vision

Parlando di robot, questi sistemi eterogenei software e hardware sono diventati sempre più pervasivi nell'ambito umano, sia da un punto di vista di utilità, si pensi solamente a quelli adibiti alla pulizia in maniera autonoma, ma anche dal punto di vista di semplici strumenti costruiti allo scopo di divertirsi e imparare che hanno portato alla generazione di un vero e proprio businnes. Quello sul quale ci si vuole concentrare è l'ambito delle Internet Of Things (IoT) che è un settore tutt'ora in espansione e per il quale, per fortuna, ancora non si conoscono limiti di utilizzo. Il corso proposto, e in generale l'Università si propone di fornire una serie di basi che spaziano dal punto di vista progettuale, implementando così anche le tecniche di Learn By Doing, a quella realizzativo di sistemi che possano essere a loro volta software factory per sistemi robotici immersi nelle differenti aree dell'IoT. Il campo di applicazione scelto, appunto quello dei robot, risulta possedere delle caratteristiche di forte dinamicità dettate dagli avanzamenti tecnologici fatti negli ultimi anni che richiedono un software sempre aggiornato all'ultima versione, in grado di funzionare correttamente in ogni condizione.

Quest'ultimo si traduce nella stesura di software, sempre con meno tempo a disposizione ma, che possa essere facilmente testato e validato. Per questi motivi a supporto dell'attività didattica di sviluppo software si vogliono sperimentare delle metodologie di produzione del software gestite da software factory, così da avere sempre una base di conoscenza consistente e che permettano la modifica e il riutilizzo di codice prodotto precedentemente, così da poterlo fare in tempi brevi per poi sottoporlo a verifiche da parte del personale. Per questi motivi il team prevede l'utilizzo di **Domani Specific Language** per la produzione di codice abbattendo i tempi e i costi di produzione e gestione.

3 Goals

L'obiettivo non è solamente quello di realizzare quando descritto nella sezione delle richieste, ma prevede anche uno studio delle metodologie di realizzazione dei sistemi di questo tipo seguendo le linee guida esposte a lezione. Relativamente al problema in esame si vuole riconoscere e valutare la presenza di un abstraction gap già al termine della fase di analisi del problema, riuscendo inoltre a discriminare tra gli aspetti relativi al dominio in questione (**domain specific**) e quelli relativi alla realizzazione dell'applicazione (**application specific**), come l'ipotesi tecnologica influisca sul processo di produzione del software. Altro punto fondamentale sul quale si vuole porre attenzione è l'estensione del Domain Specific Language aziendale utilizzato così da poter costituire un patrimonio informativo comune sempre aggiornato con nuove soluzione tecnologiche adatte a nuove problematiche evidenziate. La costituzione di un prototipo funzionante risulta essere un ulteriore goal da soddisfare, questo, come già detto, risulta essere realizzato con l'utilizzo della software factory, la quale, permette una rapida e robusta prototipazione al termine della fase di analisi consentendone la presentazione al committente per la pianificazione delle successive attività di progetto e sviluppo attraverso specifico workplan. Le linee guida alle quali si decide di ispirarsi sono quelle dettate dalla metodologia di sviluppo chiamata **SCRUM**, dalla quale cercheremo di sfruttare l'approccio di generazione e gestione del software.

4 Requirements

4.1 Fase 2

Progettare un sistema software che:

- Permetta di specificare un'azione temporizzata: ovvero esplicitando direttamente la durata dell'azione all'interno comando impartito al Differential Drive Robot (DDR).
- Permetta di definire azioni interrompibili: ovvero il robot è in grado di interrrompere l'esecuzione di un comando in seguito alla ricezione un segnale di halt proveniente da una console remota.

– Permette di controllare un DDR attraverso una console remota: ovvero inviare comandi al robot il quale deve essere in grado di interpretarli ed agire di conseguenza.

5 Requirement analysis

Avendo definto i requisiti nella prima relazione[1], si specificano in maniera più approfondita delle aree che non erano state identificate:

– Definizione di una sequenza di azioni;
– Definizione di azione temporizzata;
– Definizione di azione interrompibile;
– Definizione di console remota.

5.1 Sequenza di azioni

Una sequenza di azioni viene identificata da una serie di azioni successive che dovranno essere performate dal robot.

5.2 Azione temporizzata

Comando impartito al robot che prevede nella sua definizione oltre al tipo di azione anche la sua effettiva durata.

5.3 Azione interrompibile

Azione che prevede una esecuzione con tipologia asincrona ovvero all'interno del proprio piano, una azione asincrona prevede la possibilità di essere interrotta alla ricezione di una nuova azione o di un comando di halt proveniente dalla console.

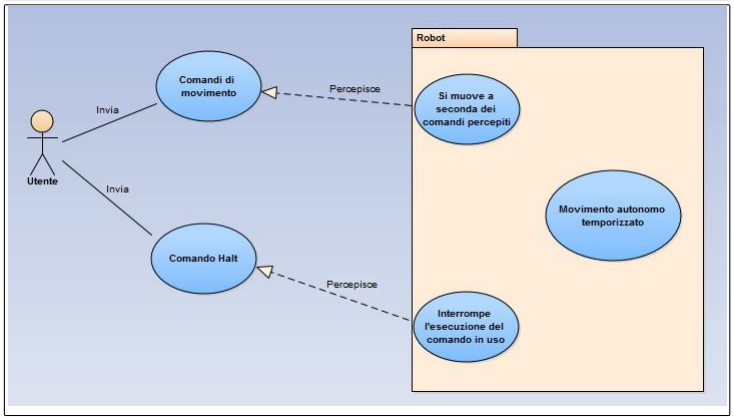
5.4 Console remota

La console remota riveste il ruolo di emittente di determinati segnali in grado di essere captati e interpretati dal robot specifico.

[1] Ingegneria dei Sistemi Software A differential drive Robot Prima Parte

5.5 Use cases

Fase 2



5.6 Scenarios

ID:	Movimento autonomo temporizzato.
Descrizione:	Il robot esegue un piano che comprende azioni temporizzate.
Attore:	Robot
Precondizione:	Il robot deve essere acceso.
Scenario Principale:	Il robot sta eseguendo i comandi previsti dal piano specificato.
Scenario Secondario:	Assenti.
Postcondizione:	Assenti.

ID:	Ricezione comando "halt".
Descrizione:	Il sistema percepisce un comando "halt" inviato da console remota e reagisce interrompendo il piano in esecuzione.
Attore:	Utente
Precondizione:	Il robot deve essere acceso e si sta muovendo in maniera autonoma.
Scenario Principale:	Il robot termina il piano in esecuzione.
Scenario Secondario:	Assenti.
Postcondizione:	Il robot è fermo.

ID:	Ricezione comandi di movimento.
Descrizione:	Il robot percepisce un comando di movimento inviato da console remota ed esegue l?azione corrispondente.
Attore:	Utente
Precondizione:	Il robot deve essere acceso.
Scenario Principale:	Il robot deve eseguire i comandi percepiti.
Scenario Secondario:	Assenti.
Postcondizione:	Il robot rimane in attesa di un comando.

5.7 (Domain)model

Si veda la relazione precedente[2].

5.8 Test plan

Si veda la relazione precedente[3].

6 Problem analysis

Visto il contesto nel quale ci si pone, le problematiche identificate hanno portato, come citato all'interno dell'analisi dei requisiti, alla definizioni di una serie di concetti per le varie modalità di esecuzione delle azioni. Occorre identificare una modalità che ci permetta di astrarre dallo specifico problema in gioco (ovvero quello dei robot) e che ci consenta di definire una soluzione generale alla problematica, sfruttano il robot come approccio, definendo i concetti validi anche per future fasi evolutive del progetto. I requisiti ci portano a riconoscere una serie di problematiche delle quali ci andremo ad occupare: definizioni di azioni sincrone/asicrone ed azioni interrompibili.
Ponendoci nel contesto di esempio, il robot deve essere sensibile ad alcuni cambiamenti che potranno avvenire nell'ambiente circostante nel quale è situato. Occorre, quindi, definire la gestione di questi cambiamenti in modo che durante l'esecuzione delle azioni previste, nel caso di rilevamento di un cambiamento, il robot possa reagire di conseguenza. In particolare si vuole fare in modo che l'azione intrapresa dal robot si blocchi e si possano eseguire azioni alternative, al termine di queste sarà necessario valutare se continuare o meno con l'esecuzione precedente che include le azioni già pianificate.
Così facendo occorre definire, dato un robot, tutti i cambiamenti ai quali dev'essere in grado di reagire, descrivendo i comportamenti da avere in ogni situazione e le modalità di controllo che permettano di fare una valutazione sullo stato di avanzamento dell'azione corrente intrapresa dal robot. Inoltre è necessario riconoscere quando un'azione è giunta al termine in modo da controllare lo stato

[2] Ingegneria dei Sistemi Software A differential drive Robot Prima Parte
[3] Ingegneria dei Sistemi Software A differential drive Robot Prima Parte

dell'azione ed eventualmente proseguire con la successiva.
La base di partenza per noi è l'astrazione di classe BaseRobot il quale tramite apposito metodo ha la possibilità di eseguire azioni tramite l'uso di appositi attuatori, all?interno del mondo reale. Legata alla problematica identificata, sorge inoltre il problema di specificare la tipologia di azione da utilizzare all'interno al contesto in esame dato che questa, avrà effetti sul controllo del robot stesso e sul comportamento dell'architettura logica creata a valle.
Si vogliono fissare le definizioni relative alle possibili azioni utilizzate all'interno del contesto del problema appena definito:

- **Azione sincrona**: si considera un'esecuzione sincrona, di una azione, quando questa viene concretizzata e occorre attendere la terminazione della stessa per poter restituire il controllo al chiamante.
- **Azione asincrona**: si definisce un'esecuzione asincrona quando una determinata azione può essere eseguita senza che il chiamante debba attendere la fine dell'esecuzione dato che il controllo viene immediatamente restituito al chiamante.

Rispetto alla fase precedente si deve considerare un sistema non più concentrato ma distribuito, infatti l'utilizzo della console remota rende necessaria la definizione della comunicazione tra quest'ultima e il robot. Dev'essere definito un comando di "halt" che la console sia in grado di inviare al robot e quest'ultimo dovrà essere in grado di interpretarlo.

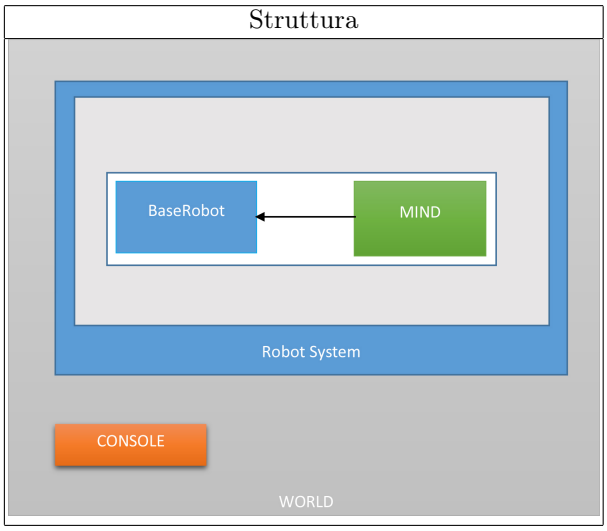
6.1 Logic architecture

Ci poniamo nell'ottica di ottenere un modello del sistema funzionante e globalmente accettato da tutte le parti in gioco, identificando i macro sottosistemi senza specificare nulla più di quanto già detto precedentemente. Il tutto rimanendo indipendenti dalla specifica tecnologia che si utilizzerà e demandando queste decisioni solo durante la fase di progetto.

Struttura Da quanto detto, si identificano principalmente tre sottosistemi:

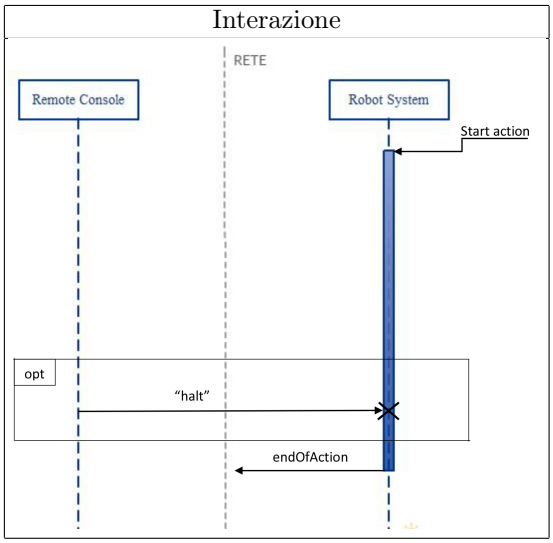
- altri eventuali sistemi in grado di ricevere segnali presenti nell'ambiente (world);
- console remota;
- robot system.

Mentre con il primo si identifica l'**ecosistema** in cui opera il robot ed altri eventuali sistemi esterni presenti, la console remota riveste il ruolo di emittente di determinati segnali in grado di essere captati dal robot system specifico o da una serie di robot interessati ad una specifica tipologia. Quest'ultimo risulta essere strutturato su una serie di livelli che vanno ad arricchire le caratteristiche del robot stesso, raggiungendo così un livello di astrazione più consono al progetto richiesto.

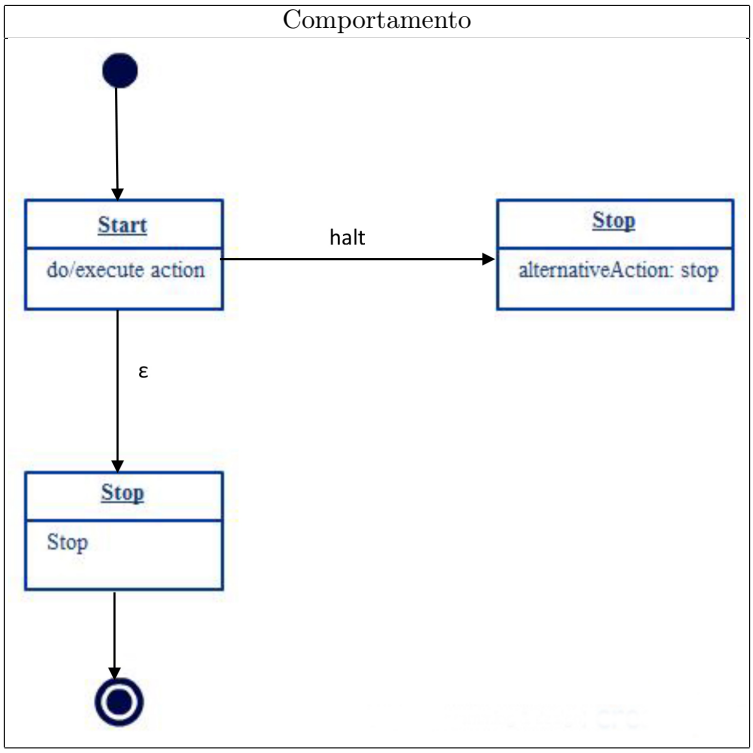


All'interno del diagramma si è voluto anche rappresentare il **"mondo"**, situando così il robot ed identificando il contesto di esecuzione del piano stesso. Un'ulteriore sottosistema identificato, come già detto, è la **console remota** che, ai fini dettati dal testing del sistema, verrà posta all'interno del sistema RemoteConsole e sarà quindi in grado di inviare il messaggio di "halt". Si lascia inoltre spazio, in questa rappresentazione, ad ulteriori sistemi presenti nel "mondo" ed in grado di interagire con i principali sistemi proposti nell'architettura logica, in modo da focalizzarsi solo sulla definizione dei vincoli di interazione e senza complicare ulteriormente la struttura e il comportamento del sistema nel suo complesso.

Interazione Ricordando la base di partenza ovvero il contesto distribuito del robot, occorre introdurre il concetto di rete come mezzo di comunicazione per l'invio di comandi. Come si può vedere dal diagramma sottostante il RobotSystem interagisce con il sottosistema RemoteConsole e nel caso in cui quest'ultimo invii un comando di "halt", il RobotSystem lo gestirà terminando l'azione che stava eseguendo. Qualora dal sottosistema RemoteConsole non arrivasse un comando di "halt", il RobotSystem inizierà la sua azione e la porterà a compimento.



Comportamento Viste le premesse fatte negli capitoli precedenti, e utilizzando la base di conoscenza a nostra disposizione proveniente da specifici corsi di studi, la problematica identificata ci spinge a modellare il comportamento del robot come un automa a stati finiti. Un primo prototipo potrebbe essere il seguente in cui viene rappresentata l?esecuzione di un azione che, nel caso in cui non avvengo la ricezione del comando "halt", termina.

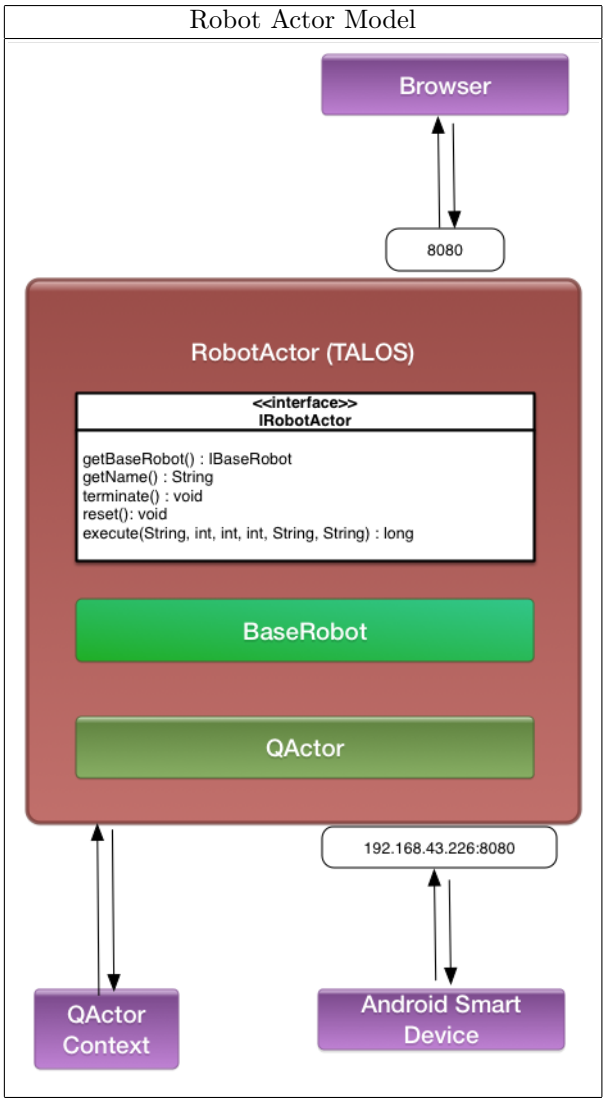


Per modellare il "comportamento di default" del robot, quindi l'esecuzione dell'azione senza l'interruzione proveniente dalla console remota, andrebbe introdotto il concetto di "epsilon mossa". Per epsilon mossa di intende una transizione che avviene senza avere nulla in ingresso, che è proprietà specifica degli automi a stati finiti. Questa soluzione non è modellabile all'interno del dominio applicativo fin qui definito, infatti supponendo di utilizzare un azione asincrona, questa terminerà immediatamente senza che venga effettivamente attuata nel mondo. Nasce quindi la necessità di introdurre un livello di astrazione superiore che permetta di esprimere in modo corretto il comportamento del robot e che consenta di effettuare la transizione di stato solo al momento opportuno, ovvero al termine dell'esecuzione dell'azione.

6.2 Abstraction gap

Avendo appena definito i concetti principali nei capitoli precedenti, ci si rende conto che la base di partenza considerata, ovvero il linguaggio OO Java, non permette di esprimere ad un sufficiente livello di astrazione le tematiche sopra citate e il dislivello dal punto di vista tecnologico sarebbe troppo ampio da essere colmato. Si procede quindi alla definizione di una serie di livelli di astrazione che permettano di arrivare gradualmente a definire le modalità di approccio al problema.

Il primo che occorre è la possibilità di una gestione tramite gli attori in quando il modello computazionale che si rifà ad uno stile proattivo/reattivo riesce a modellare perfettamente il comportamento di un robot. Questo modello in realtà risiede già all'interno del framework computazione che utilizzeremo (qActor) e potrà quindi essere considerato come assodato per le future implementazioni del problema. La base di partenza, risulta essere quindi il RobotActor, come già definito nella precedente relazione[4], esteso con la possibilità di ricevere comandi inviati dalla console remota oppure dal browser.



[4] Ingegneria dei Sistemi Software A differential drive Robot Prima Parte

E' stato detto in precedenza che per modellare correttamente il comportamento del robot occorre introdurre il concetto di automa a stati finiti, questo ci induce a passare dalla programmazione message passing quale Java a quella event based. Questo ci permette di definire che il cambiamento che permette di eseguire la transizione da uno stato all'altro dell'automa può essere identificato come un evento. In questo modo si vuole ottenere una entità reattiva che nel momento in cui esegue un?azione possa comunque essere sensibile a determinati "cambiamenti". Infatti, supponendo di essere in uno stato in cui viene eseguita una determinata azione, il robot si mantiene reattivo e nel momento in cui riceve un particolare evento è in grado transitare in un determinato stato che permetta la gestione dell?evento stesso.
In particolare, con questa modellazione, si è in grado di definire un evento che viene lanciato al termine dell'esecuzione di un azione sincrona/asincrona. Ovviamente per un particolare stato possono essere definiti più eventi ai quali il robot dev'essere sensibile ed altrettante transizioni di stato che permettano di gestirli. In un contesto generale, quando viene ricevuto un evento di halt, che provoca una transizione di stato, esso viene gestito eseguendo un piano alternativo e fermando l'azione eseguita in precedenza. Nel caso del contesto specifico questo non è necessario in quanto, l'esecuzione di una nuova azione da parte del robot, interrompe/termina automaticamente quella precedente.

6.3 Risk analysis

Dato l'ambito nel quale si colloca il progetto, ovvero distribuito ed eterogeneo, occorre considerare il fatto che all'interno dei vari nodi che costituiscono la rete di robot, che possono interagire nell'ambiente, vengono propagati un numero considerevole di eventi.
Occorre trovare un metodo per agevolare la diffusione dei vari eventi agli agenti che sono veramente interessati a riceverli, per fare questo è possibile pensare ad una modalità di identificazione degli eventi a seconda del contesto di provenienza e ad un'infrastruttura al di sotto che dovrebbe essere in grado di filtrare la propagazione degli eventi in base al loro identificativo. Un possibile esempio potrebbe prevedere una categorizzazione tra eventi locali quindi dello stesso nodo ed eventi proveniente da contesti esterni. In una prima fase di testing il robot potrà essere reattivo ai soli eventi locali mentre gli eventi provenienti da altri contesti potranno essere utilizzati in future implementazioni di robot cooperativi.

7 Work plan

A fronte di quanto richiesto nei requisiti risulta fino dall'inizio del processo la necessita? di dividere lo sviluppo del sistema in macro-iterazioni, nella precedente relazione è stata presentata la risoluzione alla problematica di esecuzione di

azioni da parte di un robot, mentre in questa si presentano le azioni che hanno portato alla creazione di una infrastruttura software che permetta ad un robot l'esecuzione di un piano rimanendo comunque reattivo a stimoli esterni come allarmi o comandi inviati dall'utente.
Il Piano di lavoro risulta essere sostanzialmente il medesimo di quello presentato precedentemente: infatti si prevede la fase di analisi dei requisiti che porta a produrre un modello del dominio non ambiguo e condiviso con il committente e con le parti in gioco. Successivamente si andra? a svolgere un'analisi dei problemi evidenziati, avente come obiettivo quello di produrre un'architettura logica che possa essere resiliente anche a problematiche non direttamente collegate con il problema in questione, producendo quindi una evoluzione della architettura logica precende.

8 Project

Il progetto risulta essere così suddiviso:

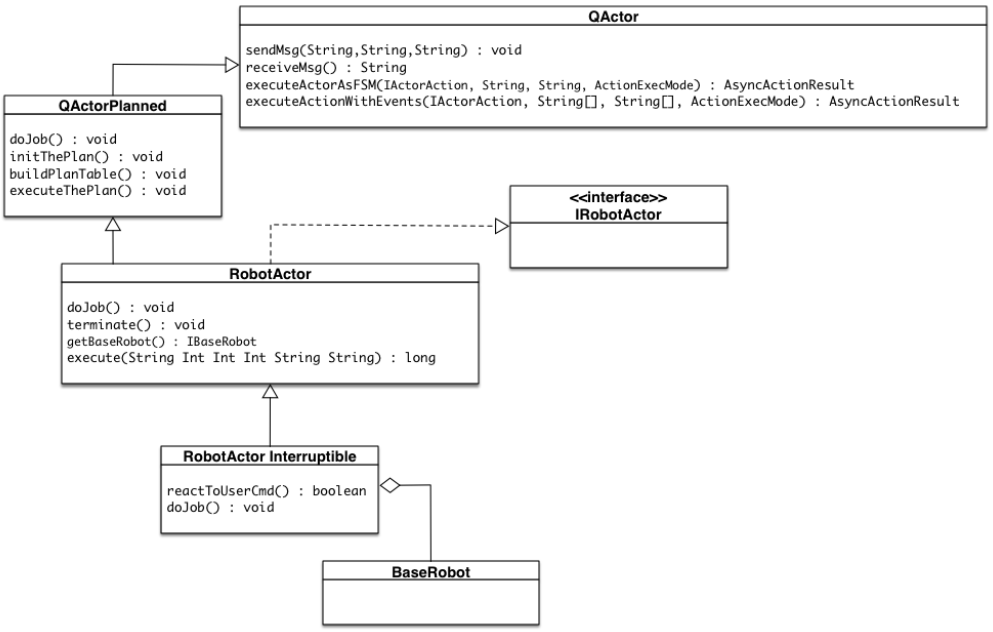
- **it.unibo.qactor.robot.test**: all'interno si possono trovare due file java. **RobotActorInterruptible.java** il quale contine la definizione del piano di esempio, nel nostro caso i comandi eseguiti dal robot saranno: forward, left, forward; e la classe **CtxRobotInterruptible** che risulta essere l'entry point del sistema.

Per quanto rigurada il codice riportato di seguito, si riferisce al piano di esempio costituito dalle seguenti azioni:
FORWARD → LEFT → FORWARD

8.1 Structure

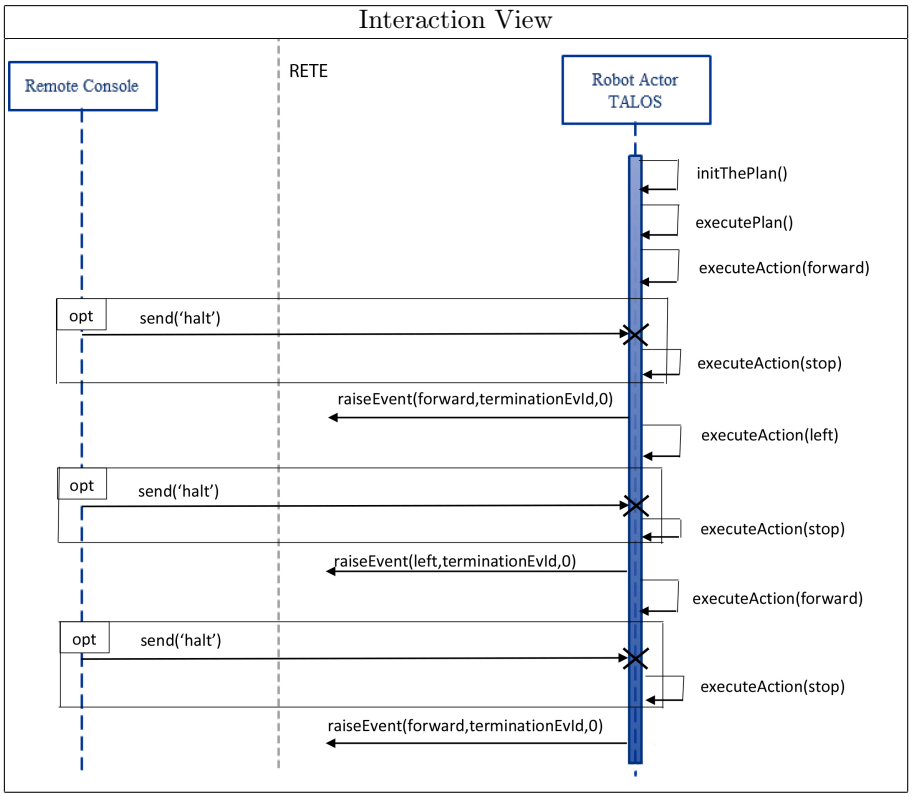
La struttura delle classi del progetto viene di seguito riportata. Per consentire una più facile consultazione sono stati riportati solo i metodi considerati principali. Inoltre non si espande la struttura della classe BaseRobot essendo già presente il modello all'interno della sezione dominio applicativo della relazione.

Structural View



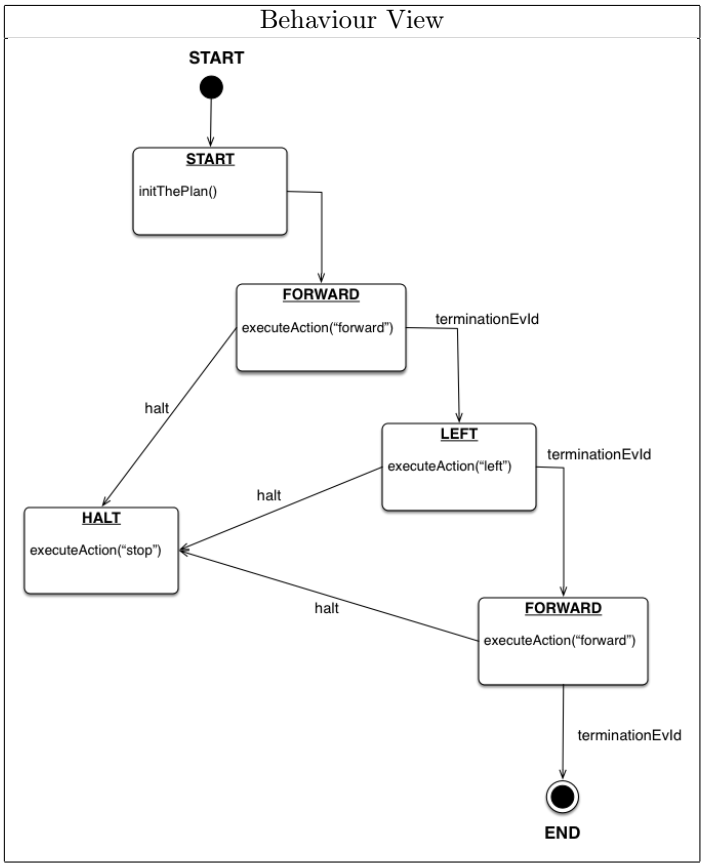
8.2 Interaction

Anche per quanto riguarda l'interazione, si prende come esempio il piano precedentemente definito e si riporta lo stack di chiamate che avvengono nel normale funzionamento del sistema, riportando inoltre i possibili comandi che possono avvenire dalla console remota.



8.3 Behaviour

Il comportamento del sistema viene, per rendere una più facile lettura, legato ad un esempio di implementazione il cui codice sarà presentato nella apposita sezione.



L'esecuzione del piano viene affidata alla funzione **executeActionAsFSM()** che crea un automa a stati finiti, come precedentemente indicato, che permette di gestire tutti gli eventuali eventi che possono arrivare durante l'esecuzione di un'azione e i possibili piani da attuare. In questo caso l'unico evento che dev'essere gestito è quello di "halt" che interrompe l'esecuzione del piano corrente e ferma il movimento del robot. E' inoltre prevista la possibilità di riprendere il piano da dove era stato interrotto, in questo caso viene fatta una distinzione tra:

- **Azione compensabile**: definita come azione che dopo essere stata interrotta, permetta l'esecuzione di un'altra azione che la compensa o la completa;
- **Azione non compensabile**: definita come un'azione che, a seguito di un'interruzione, non può essere ripristinata o completata (es. l'esecuzione di un suono).

Per poter eseguire azioni sincrone/asincrone che abbiano la possiblità di essere interrotte viene modellata l'entità TimedAction che ha la caratteristica di svolgere l'azione e emettere un evento al termine dell'azione stessa. L'utilizzo di un

ID per ogni azione permette di specificare a quale azione si rivolge lo specifico evento di terminazione. L'evento lanciato è univocamente identificato con un ID in modo da poter essere associato ad una determinata azione, in caso contrario, avendo un evento generico e supponendo di avere una sequenza di azioni "forward", l'evento bloccherà l'azione a prescindere da quella realmente terminata. Tutti gli eventi generati su un nodo vengono propagati verso gli altri nodi, per questo alcuni eventi sono definiti come "locali" ovvero che non necessitano di essere propagati sulla rete (es. evento di terminazione azione). Nel contesto del problema le azioni eseguite dal robot sono eseguite in modalità sincrona, infatti, nel caso in cui si fosse utilizzata la modalità asincrona il robot non avrebbe il tempo di attuare l'azione in quanto sarebbe subito inviato l'evento di terminazione e si passerebbe all?azione seguente.

9 Implementation

Di seguito si presenta il codice delle classi citate nella sezione di progetto.

RobotActorInterruptible.java

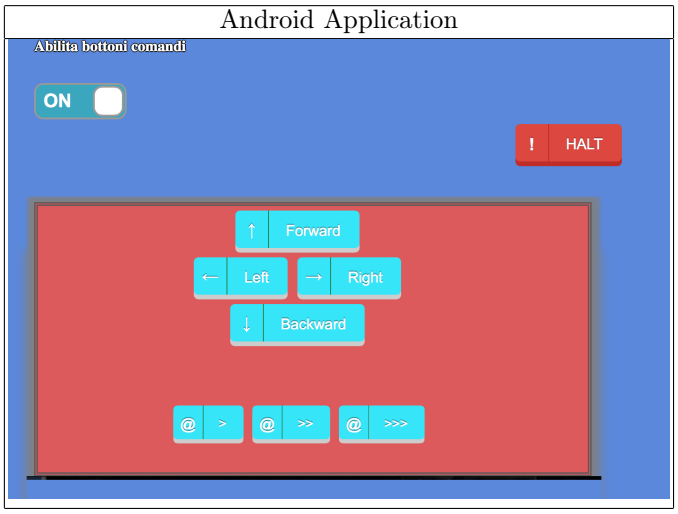
```
1  package it.unibo.qactor.robot.test.interruptible;
2  import it.unibo.iot.executors.baseRobot.IBaseRobot;
11 public class RobotActorInterruptible extends RobotActor{
12     public RobotActorInterruptible( String id, ActorContext myCtx, String planFilePath,
13             IOutputEnvView outView,  IBaseRobot baseRobot, String defaultPlan ) throws Exception{
14         super( id,myCtx,planFilePath,outView, baseRobot, defaultPlan  );
15     }
16     @Override
17     protected void doJob() throws Exception {
18         long timeforward = 2000;
19         long timebackward = 2000;
20         long timeleft = 1000;
21         println("======================================================================");
22         println("START");
23         println("GO FORWARD!");
24         do{
25             AsynchActionResult aar = execute(RobotSysKb.forwardCommand, 75, 0, (int) timeforward, GuiUiKb.terminalCmd, "reactToUserCmd");
26             if( aar.getInterrupted() ){
27                 println("interrupted");
28                 break;
29             }
30             timeforward = aar.getTimeRemained();
31         }while(timeforward>0);
32             println("GO LEFT!");
33         do{
34             AsynchActionResult aar = execute(RobotSysKb.leftCommand, 75, 0, (int) timeleft, GuiUiKb.terminalCmd, "reactToUserCmd");
35             if( aar.getInterrupted() ){
36                 println("interrupted");
37                 break;
38             }
39             timeleft = aar.getTimeRemained();
40         }while(timeleft>0);
41         println("GO FORWARD!");
42         do{
43             AsynchActionResult aar = execute(RobotSysKb.forwardCommand, 75, 0, (int) timebackward, GuiUiKb.terminalCmd, "reactToUserCmd");
44             if( aar.getInterrupted() ){
45                 println("interrupted");
46                 break;
47             }
48             timebackward = aar.getTimeRemained();
49         }while(timebackward>0);
50         println("END");
51         println("======================================================================");
52     }
54     public boolean reactToUserCmd() {
55         println(getName() + " ==== reactToUserCmd called by reflection.");
56         try {
57             this.playSound("./audio/tada2.wav", 2000, ActionExecMode.synch);
58         } catch (Exception e) {
59             e.printStackTrace();
60         }
61         return IActorAction.suspendPlan;
62     }
63 }
64
```

CtxRobotInterruptible.java

```java
package it.unibo.qactor.robot.test.interruptible;
import java.io.FileInputStream;

public class CtxRobotInterruptible extends ActorContext{

    public CtxRobotInterruptible(String name, IOutputEnvView outView,
            InputStream sysKbStream, InputStream sysRulesStream)
            throws Exception {
        super(name, outView, sysKbStream, sysRulesStream);
    }

    @Override
    public void configure() {
        try {
            IBaseRobot baseRobot = RobotSysKb.setRobotBase(this, "rbase");
            RobotActorInterruptible robot     = new RobotActorInterruptible("mock", this, "./plans.txt", outEnvView,baseRobot, "init");
            //we activate a server to receive user commmands
            //TODO : user commanda can be received also as messages or as events
            initThesServer();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    protected void initThesServer() throws Exception{
        //new RobotHttpServer(outView,8080).start();
        new TerminalHandlerExecutor("th", this, GuiUiKb.terminalCmd, outEnvView);
    }
    public static void main(String[] args) throws Exception{
        InputStream sysKbStream    = new FileInputStream("robotNaiveKb.pl");
        InputStream sysRulesStream = new FileInputStream("sysRules.pl");
        new CtxRobotInterruptible("ctxrobot", SituatedSysKb.standardOutEnvView, sysKbStream, sysRulesStream ).configure();
    }

}
```

Si riportano inoltre una serie di schermate dell'applicativo console il quale viene utilizzato come controllo per il robot da terminare Android.

Android Application



10 Testing

La fase di testing delle nuove funzionalità è stata effettuata testando prima il sistema mock ovvero virtualizzato mentre una seconda fase, quella relativa al progetto, viene testata con lo stesso robot fisico (Talos) tramite il quale erano stati effettuati i test nella prima parte.

11 Deployment

Il processo produttivo affrontato ha come fase terminale la gestione del deployment applicativo. Si prevede di distribuire l'applicazione finale mediante l'utilizzo di un pacchetto jar che dovrà essere eseguito sul robot, in particolare sul RaspBerry il quale deve inoltre essere connesso tramite apposito toogle wifi alla rete denominata natspot che viene utilizzata come mezzo trasmissivo dei messaggi e degli eventi da e per il robot. Si prevede la distribuzione dell'applicazione android sottoforma di un archivio con estensione APK, essendo il formato standard per la distribuzione delle applicazioni di quel tipo.

12 Maintenance

Ci si rende conto che il progetto realizzato presenta una serie di problematiche, identificate nella fase di analisi del problema, che sono comuni a progetti inseriti nell'ambito della programmazione distribuita ed eterogenea. Si è cercato di risolvere le problematiche presentate ad un livello generale così da poter inglobare concetti anche non direttamente correlati alle richieste ma che sono comunque

legate all'ambito. Arrivati a questo punto, si vuole consolidare in maniera sicura il know-how sviluppato riguardo al problema proposto, utilizzandolo come base salda per futuri progetti ed evitando così di porsi nuovamente problemi per situazioni già affrontate in precedenza evitando così di riscrivere parti di codice.
In tutto questo, l'aiuto fornito dall'infrastruttura dei QActor e RobotActor risulta essere molto valido e si vuole, con l'esperienza maturata, spostarsi su un modello implementativo del tipo Model Driven Developer che permetta la costruzione di modelli che si basino su quanto già appreso e creato in precedenza, spingendo per una fase di modellazione che porti alla generazione di un prototipo funzionante nel minore tempo possibile avendo una base di conoscenza del problema alle spalle ampia e solida.
Le modalità a questo punto sono due: la prima prevede di creare una serie di plugin, legati al linguaggio fino a qui scelto, che dovranno essere distribuiti in ogni area di sviluppo aziendale; questo non risulta essere un approccio comodo nel momento in cui occorrerà affrontare le problematiche di upgrade al codice oppure. Quella più plausibile, prevede l'utilizzo dei concetti definiti per le problematiche anticipate e spostarli all'interno di un Domain Specific Language cosi da poter produrre già a livello di analisi dei prototipi funzionanti e demandare al livello progettuale solo la definizione di caratteristiche tecniche. In questo modo inoltre si risolve la problematica legata alla scelta della tipologia di linguaggio o di paradigma utile ai fini del progetto, rompendo i vincoli utilizzando un metamodello custom personalizzabile secondo le nuove esigenze che potranno essere incontrate.

13 Information about the author

Alessia Papini	Beatrice Mezzapesa	Lorenzo Pontellini