Ingegneria dei Sistemi Software: A Differential Drive Robot System

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

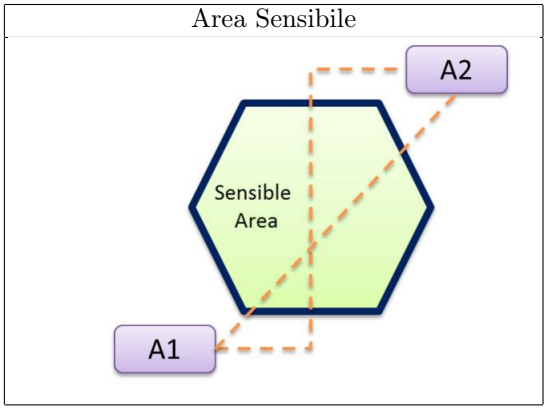
Parlando di robot, questi sistemi eterogenei software e hardware sono diventati sempre più pervasivi nell'ambito umano, sia da un punto di vista di utilità, si pensi solamente a quelli adibiti alla pulizia in maniera autonoma, ma anche dal punto di vista di semplici strumenti costruiti allo scopo di 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 utilizzi. 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 questo 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 permetta la modifica e il riutilizzo (di codice prodotto in una versione precedente) alla base di tutto di modo tale poterlo modificare in tempi brevi e sottoporlo a verifiche da parte di personale opportuno. 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'obbiettivo 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**), arrivando alla conclusione di 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 1

Progettare un sistema software che:

- Permetta ad un Differential Drive Robot, **DDR** di muoversi in maniera autonoma, da un punto prefissato A1, ad un punto prefissato A2, considerando un insieme di assunzioni sull'ambiente di azione del robot (nessun dislivello, nessun ostacolo).
- Emetta un segnale chiamato <**RobotName**> **Enter** quando il robot entra all'interno di un'area sensibile, **"Sensible Area"**, delimitata da una linea nera, ed emetta un segnale chiamato <**RobotName**> **Exit** quando il robot esce dall'area sensibile.
- permetta al robot di reagire (il prima possibile) ad uno specifico comando di **halt** inviato da un utente (attraverso una console remota).

4.2 Fase 2

Dopo aver sviluppato un prototipo, consentire la possibiltà di aggiungere funzionalità al robot permettendogli di:

- percepire un ostacolo all'interno della Sensible Area e una volta individuato l'ostacolo:
  - eseguire un comportamento alternativo;
  - (opzionalmente) usare una webcam per scattare una foto all?ostacolo ed inviarla ad una qualche user command console e/o usare qualche dispositivo per emettere un suono o un alert vocale.

5 Requirement analysis

Come specificato da requisiti, si identifica in maniera chiara l'ambito in cui si può insediare il sistema ovvero la gestione di robot. Da conoscenze precedentemente acquisite[1], si cerca di seguito di specificare le caratteristiche fondamentali dell'ambito in questione.
Un robot è un sistema autonomo che esiste nel mondo fisico, e per questo soggetto alle leggi fisiche più comuni (gravità, perturbazioni magnetiche, ecc..). Esso è in grado, se opportunamente equipaggiato di percepire l'ambiente e reagire portando a termire dei compiti. Le parti che costituiscono un robot sono:

- **Corpo fisico**: così da poter operare e sorreggersi;
- **Sensori**: componente Hardware e Software in grado di rendere il robot abile a percepire un determinato stimolo presente nell'ambiente;
- **Effettori**: componente fisico che realizza l?attuatore;
- **Attuatore**: dispositivo Hardware che realizza un qualche compito;
- **Controller**: parte Software del robot che gli permette di essere autonomo.

Il comportamento del robot non è semplicemente identificato dallo specifico controller programmato, ma è una interazione completa tra le parti che costituiscono il robot stesso.

Vengono riportate ora una serie di caratteristiche identificative dei robot. Un robot può essere definito inoltre come agente autonomo intellligente quando possiede le caratteristiche di sopravvivre in un ambiente complesso. Nel caso in cui non sia necessario l?intervento umano, l?agente è definito completo.

- **SELF SUFFICENCY**
  Abilità di sussistenza del robot verso se stesso, in un periodo di tempo prolungato
- **AUTONOMIA**
  Libertà da controlli esterni, l'autosufficenza incrementa il livello di autonomia. Le modalità per cui un agente può controllarne un altro dipende dal livello di conoscenza dello stato, dei meccanismi interni dell'agente che deve essere controllato. Si ha maggiore autonomia se si è in grado di apprendere.
- **SITUADNESS**
  Un agente è situato se acquisice informazioni riguardo l'ambiente in cui vive solo grazie all'utilizzo di sensori che interagiscono con lo stesso. Un agente situato interagisce con il mondo senza l'intervento umano.
- **EMBODIEMENT**
  Gli agenti vivono in un ambiente fisico e non si può prescindere da quello, questo ha influenza anche sulle caratteristiche che deve avere il robot e i compiti che è in grado di eseguire. Si vuole il più possibile sfruttare la relazione che esiste tra caratteristiche dell'ambiete e quelle del robot stesso.
- **ADATTATIVITA?**
  Abilità di ?adattarsi? di un robot alle caratteristiche dell'ambiente. L'agente presenta la stessa struttura a fronte di cambiamenti di condizioni dell'ambiente.

[1] Corso: Sistemi Intelligenti Robotici

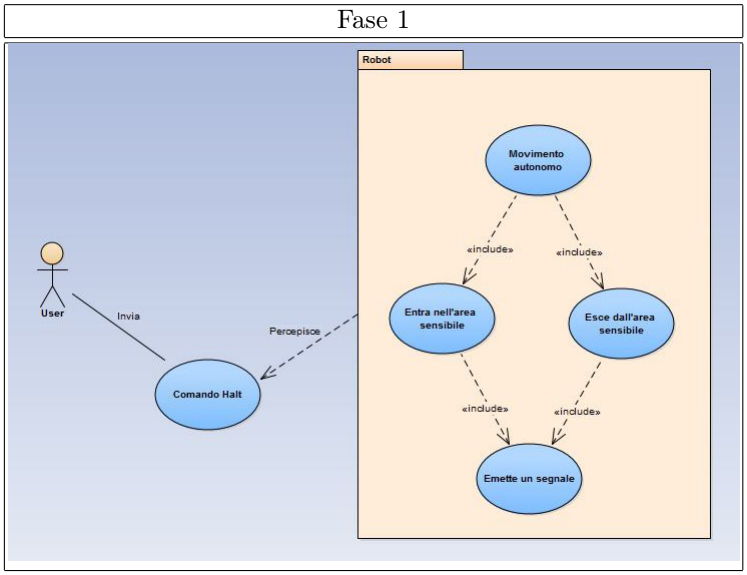
Dopo questa breve descrizione di alcune caratteristiche che identificano i robot, ci si concentra sui requisiti definiti dall'utente.

Si nota come si possa identificare sicuramente la presenza di un sistema Hardware e Software eterogeneo e distribuito che appunto sarà il robot e inoltre anche la presenza di un "emettitore di segnali" che il robot possa essere in grado di percepire e agire di conseguenza, la console. Come si può intuire dalle specifiche l'area di interesse è quella dei robot.

Possiamo definire una serie di meta-requisiti relativamente all?area di interesse:

- Possiamo dire che un Differential Drive Robot è una entità in grado di eseguire una serie di comandi di movimento, il comando può essere richiamato da un elemento esterno al robot chiamato Mind.
- Il componente Mind può essere messo in eseguzione sullo stesso supporto computazionale del robot o su di uno a parte. Nel primo caso diremo che il robot è controllato in maniera **Embedded-mode**, mentre nel secondo caso diremo che il robot è controllato in **Avatar-mode**.
- Una sequenza di comandi di movimento emesse dalla Mind è chiamata Piano.

5.1 Use cases



5.2 Scenarios

Nome:	Movimento Autonomo
Attore	..
Descrizione:	..
Trigger:	..
Precondizione:	..
Scenario Principale:	..
Scenario Alternativo:	..
Postcondizione:	..

Nome:	Emissione segnale
Attore	..
Descrizione:	..
Trigger:	..
Precondizione:	..
Scenario Principale:	..
Scenario Alternativo:	..
Postcondizione:	..

Nome:	Entrata in Area Sensibile
Attore	..
Descrizione:	..
Trigger:	..
Precondizione:	..
Scenario Principale:	..
Scenario Alternativo:	..
Postcondizione:	..

Nome:	Uscita dall'area sensibile
Attore	..
Descrizione:	..
Trigger:	..
Precondizione:	..
Scenario Principale:	..
Scenario Alternativo:	..
Postcondizione:	..

Nome:	Ricezione comando halt
Attore	..
Descrizione:	..
Trigger:	..
Precondizione:	..
Scenario Principale:	..
Scenario Alternativo:	..
Postcondizione:	..

5.3 (Domain)model

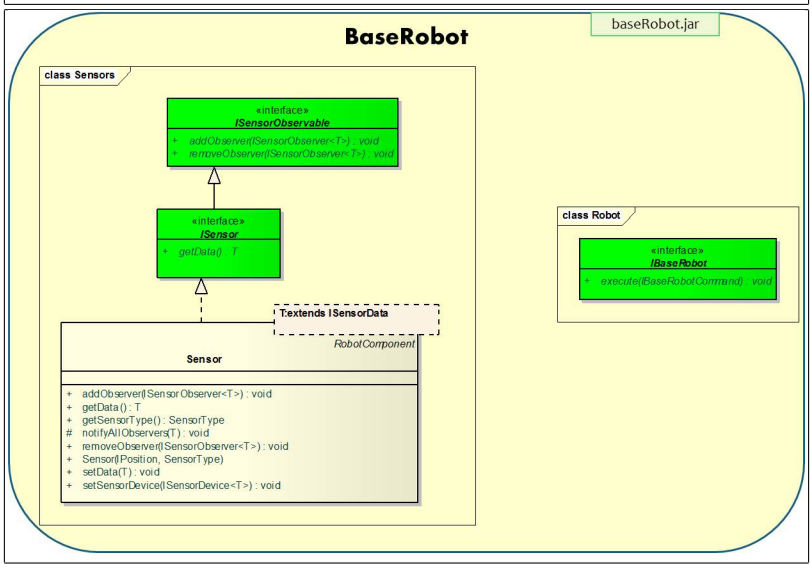
In questa sezione viene analizzato il dominio fornito dall'attuale know-how aziendale. Esso viene descritto tramite una analisi incentrata sulle tre dimensioni fondamentali caratteristiche di analisi del software[2] ovvero struttura, interazione e comportamento. Si identificano i seguenti componenti le cui caratteristiche saranno specificate in sezioni apposite che vengono utilizzate come base di partenza per la realizzazione del sistema:

- BaseRobot;
- Sensore;
- Mossa/Comando.

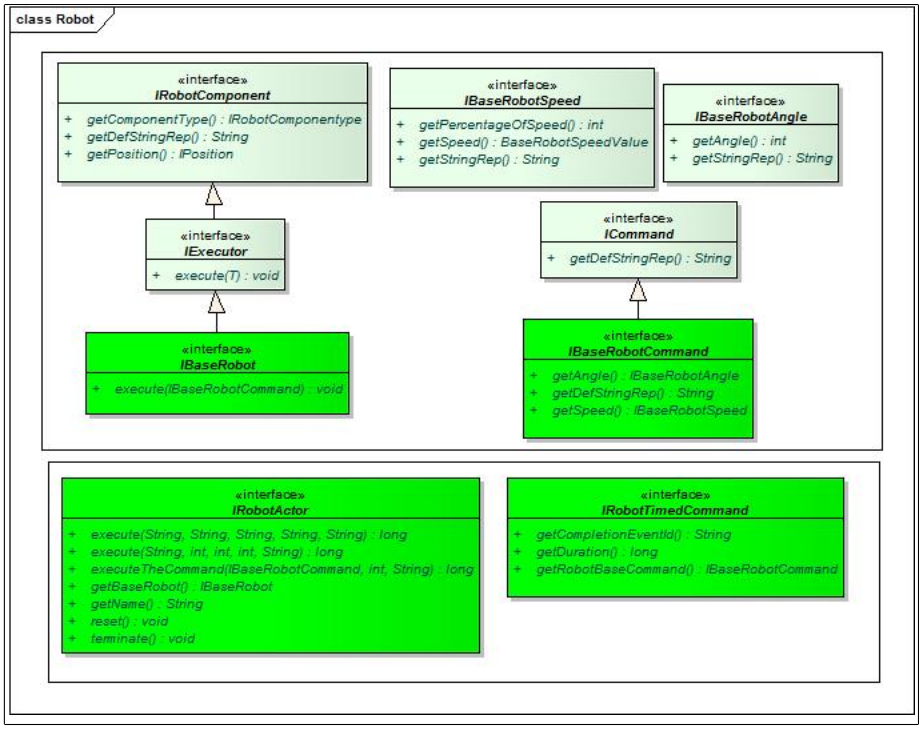
BaseRobot Un BaseRobot è un componente POJO (Plain Old Java Object) ovvero un componente Java non legato ad alcuna restrizione diversa da quelle dovute dalle specifiche del linguaggio Java stesso, che può essere istanziato per eseguire una serie di mosse. Esso rappresenta appunto l'astrazione di un robot fisico in grado di eseguire mosse e costituito da una serie di attuatori. Nulla più si può dire riguardo alla struttura fisica del robot stesso senza entrare troppo nel dettaglio, non essendo questa la parte di documentazione relativa al trattamento di certe specifiche. L'esecuzione di suddetti comandi avviene tramite interazione del tipo procedure-call.

[2] Corso: Ingegneria del Software

Base Robot

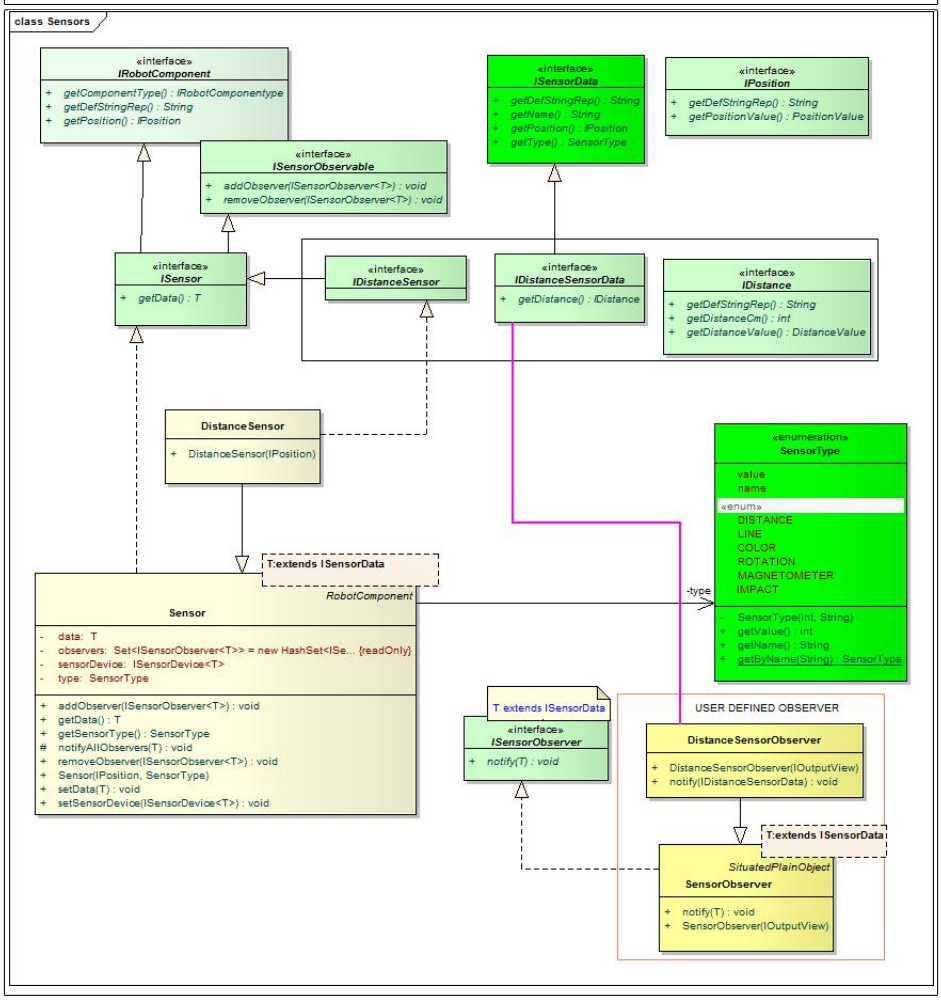


Robot Models



Sensori Il BaseRobot è equipaggiato on-board con una serie di sensori sempre del tipo POJO. Dal punto di vista del dominio, un sensore è un componente osservabile in grado di raccogliere e gestire dei dati, di un tipo specifico, provenienti o da un trasduttore fisicamente esistente o da un componente che simula la generazione di quel tipo di dati. Oltre alla modalità appena descritta è possibile interrogare un sensore anche in modalità polling, ovvero a richiesta. I sensori presenti sono caratterizzati da una tipologia specifica che ne definisce l'uso e le funzionalità in relazione a tutti gli altri.

Sensor Models



Mossa/Comando Un comando risulta essere un'istruzione di base per il robot che gli permette di effettuare qualcosa che ha un impatto sul mondo fisico nel

quale il robot è situato. Le interfacce fornite prevedono la definizione di comandi a differenti livelli di specializzazione.

5.4 Test plan

6 Problem analysis

Tramite la fase di analisi del modello del dominio, sono state descritte le specifiche del sistema in relazione alle richieste del committente, il successivo passo è quello di identificare il problema e svolgere una analisi dello stesso iniziando ad identificare i sotto-sistemi che compongono il sistema nella sua interezza, e i loro componenti. Il tutto ha lo scopo di fornire una architettura logica consona allo scopo identificato e di essere abbastanza flessibile da permettere di adattarsi senza troppe modifiche e problematiche, permettendo la definizione di un primo prototipo funzionante del sistema richiesto. L'ultimo tassello sarà la progettazione del sistema, prima di fare ciò però, occorre identificare l'abrastraction gap e i modi per poterlo colmare evidenziando inoltre i rischi e le modalità di fronteggiare lo stesso durante le successive fasi di sviluppo del software. Dopo aver fatto chiarezza sull'ambito di interesse del problema in questione avendo descritto il modello del dominio nel quale ci caliamo, si individuano le problematiche che ci si pongono davanti per portare a termine il progetto. Si cerca inoltre di ricondursi a problematiche ricorrenti fornendo concetti utili per affrontare le successive fasi previste di progettazione e realizzazione del prodotto finale.

6.1 Problematiche identificate

Ciò di cui disponiamo a questo livello è solo un componente POJO che realizza le caratteristiche associate ad un BaseRobot (precedentemente descritto nella sezione di analisi del dominio), da quanto studiato in Sistemi Intelligenti Robotici, e ripreso in maniera veloce nella fase di introduzione ai requisiti, il robot per funzionare ha bisogno di un componente apposito che ci permetta di controllare il robot a livello applicativo. Si vuole, rimanendo fedeli a tale definizione, disaccoppiare il controllo dall'effettivo esecutore. Questo perchè risulta una scelta vincente dal punto di vista ingegneristico della separazione dei concetti. Per questo come già accennatto si attua un modello in cui il robot viene comandato da un componente esterno che risulta essere la mind ovvero la mente di elaborazione.

Dalla definizione dei requisiti si identifica inoltre come il sistema debba essere veicolato verso un'interazione distribuita sulla rete, dato che deve poter reagire al comando di halt proveniente appunto da remoto. L'astrazione di partenza, non è in grado di esprimere questa caratteristica (comunicazione tramite rete) ed è per questo che occorre aggiungere una serie di livelli applicativi che potranno andare a colmare le mancanze identificate.

Gestione dei messaggi Ricopre un ruolo fondamentale la gestione dei messaggi i quali, come gia detto possono essere interni al robot per l'esecuzione di una determinata operazione, oppure esterni e quindi dettati da una console remota. Si riprendono di seguito dei concetti relativi alla gestione di messaggi appresi in corsi precedenti[3]:

- INTERAZIONE ASINCRONA: In questa tipologie di comunicazione si considera l'utilizzo di un buffer, senza alcuna limitazione sulla dimensione, questo fa si che l'emittente non debba attendere nessuna informazione di ritorno anche quando manda informazioni ad uno specifico destinatario. Il ricevente attende solo quando il buffer risulta essere vuoto. E' per questo che la comunicazione assume anche il nome di Bufferizzata;
- INTERAZIONE SINCRONA: In questa tipologia di comunicazione non si fa uso di alcun buffer. L'emittente e il desinatario scambiano informazione unificando concettualmente le proprie attività.

Visto il contesto nel quale ci poniamo, si considerano le comunicazioni essere asincrone.

Una ulteriore suddivisione che si può attuare sulla forma di messaggi è la caratterizzazione secondo diffenti forme di cui quelle che consideriamo sono solo due:

- REQUEST-RESPONSE: Si considera l'invio di un messaggio senza da un emittente verso un destinatario senza ottenere un messaggio di risposta di alcun tipo. Si ipotizza infatti che il messaggio arrivi ma non vengono effettuate assunzioni a riguardo.
- DISPATCH: Forma di comunicazione in cui il mittente invia un mesaggio e si aspetta un messaggio di risposta conforme a quanto richiesto con l'assunzione della effettivo soddisfacimento della richiesta.

Gestione Sensori —-RICONTROLLARE—- Da quanto appreso nel modello del dominio, i sensori collegati al robot si comportano secondo il pattern Observer, questa caratteristica non risulta essere la più consona in quanto l'utilizzo di questo pattern porta con se features non consone all'obiettivo nostro:
In questo caso vi è una stretta relazione tra sorgente del segnale e observer questa caratteristica mina la reattivita della sorgente. Essendo il robot in generale costituito da una serie di sensori e non solo uno, la gestione di numerosi observer può rallentare la generazione dei dati, anche in questo caso la colpa è da imputare al passaggio di flusso di controllo tra sorgente e observer. Dopo questa breve analisi si nota subito che una soluzione che sfrutta questa tecnologia non risulta consona al contesto verso cui si sta dirigendo il sistema. Si vuole infatti disaccoppiare il flusso di controllo.

Un altro possibile approccio è quello appreso durante quest'ultimo anno[4] che getta le basi per una gestione event-driven che permette di disaccoppiare sorgente e gestione dell'evento stesso. In questo caso il comportamento è organizzato

[3] corso: Ingegneria del Software triennale
[4] Corso: Programmazione Avanzata e Paradigmi

in un insieme di handler che si occupano di gestire gli eventi, incapsulando la computazione da eseguire alla percezione di un determinato evento. L'esecuzione della computazione asincrona è atomica e gli eventi che si verificano durante tale esecuzione sono inseriti in una coda di eventi che è utilizzata per tenere traccia degli eventi generati dall'ambiente e dagli handler stessi. Il comportamento del sistema è rappresentato da un modello di esecuzione chiamato event-loop così definito:

```
loop{
Event ev = evQueue.remove()
Handler handler = selectHandler(ev)
execute(handler)
}
```

L'event-loop è definito internamente all'infrastruttura in modo da non essere visibile agli sviluppatori che si occuperanno unicamente di specificare la selezione e l'esecuzione degli handler. Nonostante i vantaggi di questo approccio, sono emerse anche alcune problematiche alle quali ci si riferisce con il termine **callback hell**, tra queste si individua un'alta frammentazione del codice in handler asincroni e l'innestarsi di callback che incrementano la complessità del codice, a sfavore di leggibilità, riusabilità ed estensione. Questo rende decisamente difficile la programmazione e le richieste relative alla verifica dello stato del mondo. —QUI MANCA UN PEZZO CHE UNISCA I DISCORSI— Un approccio alternativo può essere definito con l'introduzione di un componente: il Task, che rappresenta un entità autonoma che lavora in modo proattivo, tale comportamento è de event-based e più precisamente un task potrebbe anche non terminare la proprio esecuzione. Io comportamento di un task è modellato come un automa a stati finiti che gestisce gli eventi in accordo con il suo stato interno.

CONCLUSIONI PARZIALI Si è ora in grado di poter descrivere un sistema robotico in maniera abbastanza completa relativamente anche alle caratteristiche complesse dell'agente. La successiva caratteristica che emerge dai requisiti è quella di autonomia relativamente alle richieste del utente. Come descritto precedentemente, si parla di autonomia di un robot quando non è prevista l'interazione con l'utente durante l'esecuzione di un compito assegnato, quindi in grado di auto-governarsi seguendo leggi e strategie predeterminate.

Il sistema in esame è in grado di darsi dei comandi e reagire in un certo modo nel momento in cui si verificano situazioni predeterminate. E' necessario definire un modello che definisca il comportamento standard del robot e che gli permetta di reagire nel momento che si verifica una determinata condizione azionando un comportamento alternativo a quello già in uso. Per queste motivazioni si introduce il concetto di piano definito come sequenza di comandi temportizzati. Durante la cui esecuzione standard, il robot è in grado di reagire a situazioni preidentificate mettendo in azione un piano alternativo. — QUI

CHE FIGURA CI VA?— In figura vengono prensentate le caratteristiche fondamentali che definiscono un piano dal punto di vista strutturale, riportando gli elementi che lo costituiscono e la relazione esistente tra questi. Avendo precedentemente definito il concetto di Task, questo risulta essere l'approccio ideale alla definizione di un piano. Questo perche il piano si integra perfettamente con una descrizione.

6.2 Logic architecture

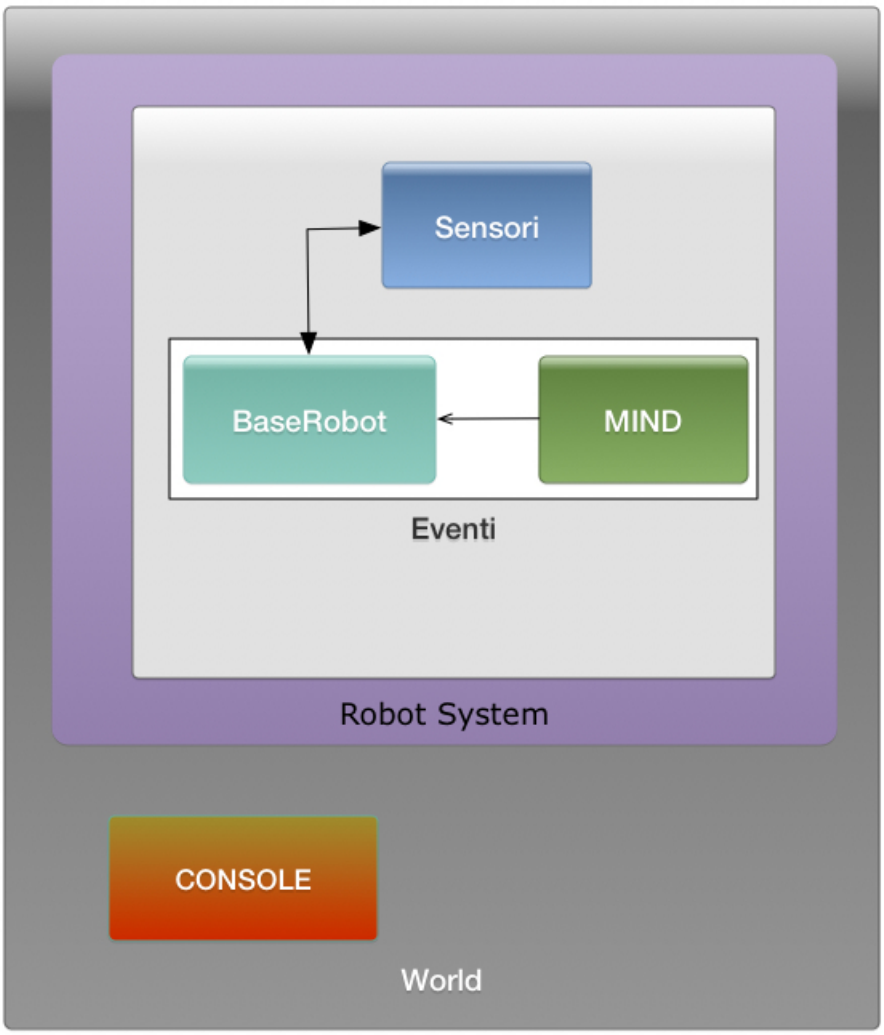
Ci poniamo nell'ottica di ottenere un modello del sistema funzionante e globalmente accettato da tutte le parti in gioco, identificando i macro sottosistemi senza specificare nulla più di quanto già detto precedentemente. Il tutto rimanendo indipendenti dalla specifica tecnologia che si utilizzerà e 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. Il primo layer, consente la gestione della comunicazione a scambio di messaggi, mentre si è deciso di sfruttare un ulteriore layer che permetta di realizzare una infrastruttura ad eventi avendo identificato, in fase di analisi, alcune caratteristiche consone per gestire al meglio le informazioni provenienti dai sensori.

Struttura



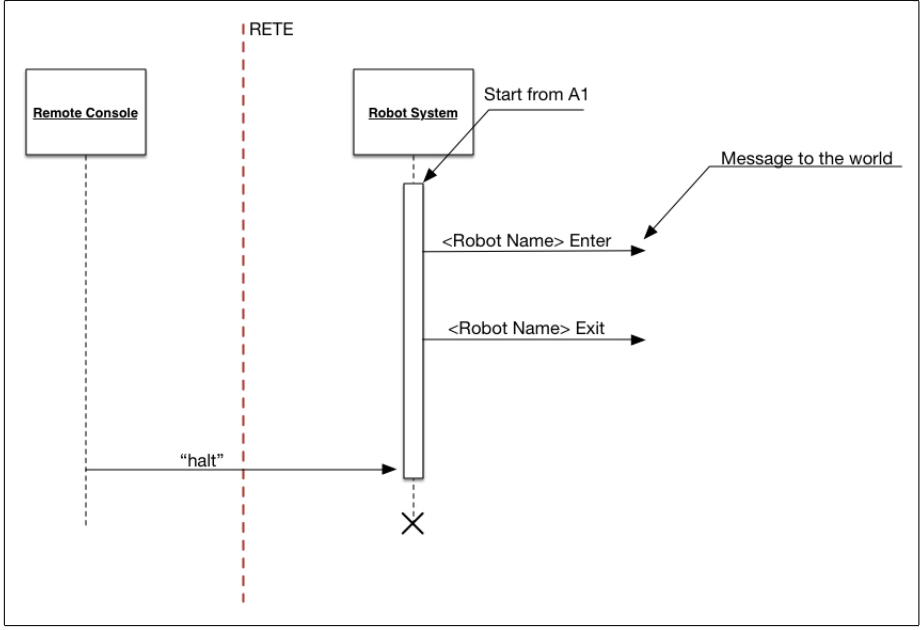
All'interno del diagramma si è voluto anche rappresentare il **"mondo"**, situando così il robot ed identificando il contesto di esecuzione del piano stesso. Un'ulteriore sottosistema identificato, come già detto, è la **console remota** che, ai fini dettati dal testing del sistema, verrà posta all'interno del sistema Remote-Console 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 Il RobotSystem può interagire sia con sistemi esterni, che possono essere rappresentati da altri robot, oppure con il sottosistema RemoteConsole, entrambe le interazione sono analizzate nel dettaglio di seguito. Il RobotSystem interagisce con i sistemi esterni inviando i segnali <**RobotName**> **Enter** e <**RobotName**> **Exit** che identificano rispettivamente l'entrata e l'uscita del robot dalla "SensibleArea". Questo segnale è rappresentato da un evento che può essere percepito da tutti i sistemi esterni interessati che si registrano alla sorgente di informazione, in questo caso il robot. La semantica dell'interazione è rappresentata dalla dispatch che prevede l'invio di un messaggio senza che il sistema resti in attesa di una risposta, la comunicazione è quindi di tipo asincrono. Il RobotSystem è inoltre in grado di ricevere un segnale di tipo "halt" inviato da una RemoteConsole. In questo caso il segnale è indirizzato e percepito da uno specifico robot e può essere gestito tramite l?invio di un messaggio con la seguente struttura:

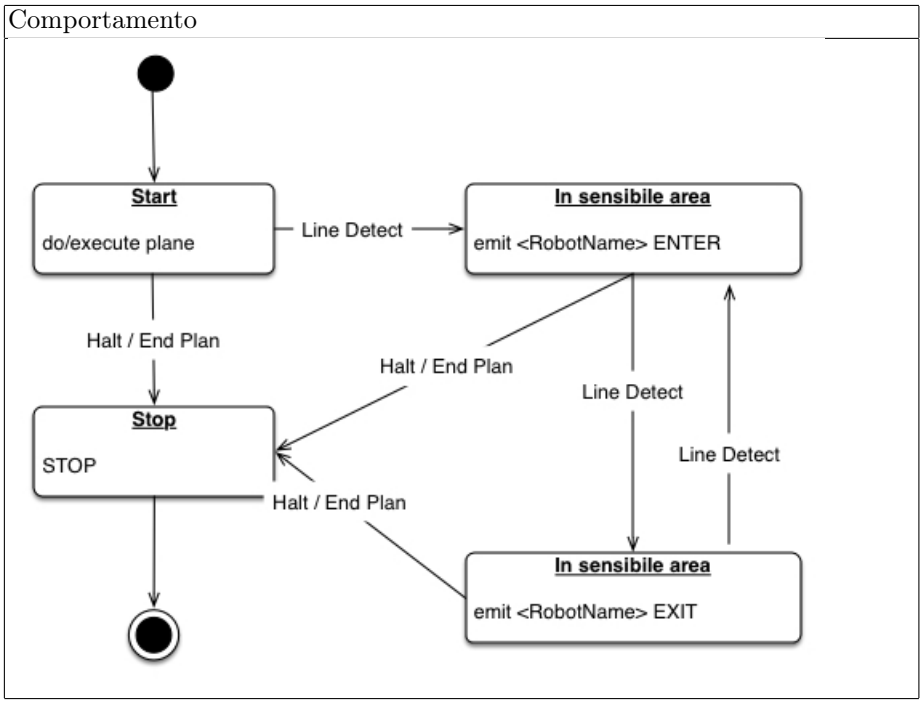
msg(MSGID, MSGTYPE, SENDER, RECEIVER, CONTENT, SEQNUM)

MSGID: è il nome che identifica il messaggio
MSGTYPE: identifica il tipo di messaggio utilizzato (dispatch)
SENDER: rappresenta colui che invia il messaggio
RECEIVER: identifica colui che deve ricevere il messaggio
CONTENT: contenuto del messaggio (halt)
SEQNUM: rappresenta un numero che viene incrementato ad ogni invio di un messaggio.

Interazione



Comportamento Una volta partito, il robot system mette in esecuzione il piano e genera l’evento di ricezione della linea, recepita utilizzando un sensore di linea, che segnala l’ingresso o l’uscita dall’area sensibile in base al fatto che la linea sia percepita una prima o seconda volta. In tutti gli stati posso giungere allo stato finale di stop sia ricevendo un "halt", inviato dalla console remota, sia un "end plan" che dichiara la fine dell?esecuzione del piano richiesto.



Per descrivere al meglio il comportamento del sistema abbiamo deciso di utilizzare un DSL (domain specific language), utile per descrivere la configurazione del robot e adatto alla prototipazione rapida. Il DSL è solo un punto di partenza, è abbastanza facile per la modifica della sintassi mentre la semantica del linguaggio è espressa dal codice generato, quest?ultimo rappresenta un modo per superare l?abstraction gap tra il problema e la tecnologia di riferimento.

6.3 Abstraction gap

Avendo a che fare con un argomento caratterizzato da una impronta tecnologica molto spinta che porta ad avere concetti innovativi, ci si rende conto che le astrazioni di base proposte con la programmazione object oriented fino ad ora utilizzata, non sono più in grado di soddisfare le problematiche fino a qui identificate. I concetti fino ad ora identificati portano un abstraction gap consistente in riferimento alla nostra ipotesi di base. Per riuscire a superare il divario di astrazione in modo "modulare" si fa riferimento ad una serie di frameworks (personalizzati) come QActors e QStream ??(per quando riguarda la gestione di messaggi), QEvents (per la gestione dell'event driven programming). Per sfruttare appieno l'uso dei framework proposti, si usa il DSL aziendale che permette di esprimere in maniera formale e intuibile da tutti il funzionamento del sistema e di ottenere in maniera automatica il codice eseguibile relativo. L'uso del DSL aziendale non viene scelto solo per la semplicità d'uso ma per la caratteristica di

permettere la generazione di codice del tutto automatica e quindi permettendo di abbattere i costi di produzione del software.

Nella figura successiva viene visualizzata la struttura layer identificata a partire dall'object oriented programming, deciso come base di partenza, a cui si appoggiano i layer precedentemente identificati, i quali potranno essere tutti sfruttati dall'ultimo ovvero highRobot.spec per colmare l'abstraction gap identificato.

6.4 Risk analysis

I rischi legati al progetto sono da imputare alla scelta fatta di utilizzare il DSL aziendale come fonte per colmare l'abstraction gap precedentemente identificato, infatti questa scelta definisce una serie di vantaggi e svantaggi.

SVANTAGGI

- Per poter comprendere appieno la semantica del linguaggio occorre investire una certa quantità di tempo nell'analisi del codice generato. Questo è dovuto al fatto che non risulta essere ancora presente un modello opportuno dei comandi, e delle operazioni computabili attraverso il DSL. Manca inoltre ancora una documentazione precisa per poter permettere anche ad utenti meno esprerti di formarsi in relazione all?utilizzo del DSL. Per questo motivo parte del tempo speso anche dal team è stato per cercare di comprendere le scelte che hanno portato il system designer a risolvere un determinato problema in un modo piuttosto che in un altro. L'unico modo per poter comprendere questo è stata un'attenta analisi del codice generato. Per questo, vi è ancora un certo grado di ambiguità per alcuni dei concetti inseriti all'interno del DSL, portando l'utente che lo sfrutta a lavorare con uno strumento di modellazione che esprime concetti non completamente chiari e condivisibili da tutti;

- Un altro elemento negativo identificato, proviene parzialmente dalle caratteristiche espresse nella parte precedente ed è inoltre legato alla struttura intrinseca del DSL stesso, ovvero la sua struttura layered. Infatti come detto prima, per poter essere padroni nell'utilizzo del linguaggio occorre una lunga fase di testing sperimentale che porta ad avere una curva di apprendimento più lunga non avendo una documentazione sulla quale appoggiarsi ne in fase di utilizzo per la generazione di codice ne nella successiva fase di debug. Inoltre se si identifica un problema all'interno di uno dei layer che costituiscono il DSL difficilmente qualcuno di diverso da chi ha scritto il codice riuscirà a risolvere la problematica identificata.
- La struttura del DSL così costituita porta a fare si che nel momento in cui si voglia introdurre un upgrade della struttura per modificare un comportamento, o di una modifica della semantica stessa, presente all'interno di uno dei layer definiti, le ripercussioni che si avranno saranno identificabilli solo in fase di messa in esecuzione del sistema e non in fase di definizione della nuova feature. La modifica inserita inoltre, data la mancanza di un modello

formale, rischia inoltre di riperquotersi in manierea impredicibile su ciò che è già stato creato in precedenza.

Dalle problematiche idenificate si nota come risulti fondamentale definire almeno un modello che chiunque possa sfruttare per capire i concetti esprimibili con il DSL e gli permetta, anche senza una conoscenza approfondita delle caratteristiche di ogni layer costituente, di sfruttare appieno le caratteristiche investendo così più tempo nella risoluzione avendo mitigato in parte la curva di apprendimento di uso del software avendo a disposizione un aiuto in più. Inoltre risulterebbe utile definire una serie di invarianti di sistema, magari concordando pareri di un system designer (chi cioè crea ed estende il DSL) con un project designer (chi sfrutta il DSL) per definire una serie di concetti che anche in un nuovo update di versione non debbano mutare.

VANTAGGI Si vogliono comunque specificare i numerosi aspetti positivi identificati nell'uso del DSL, infatti anche se sono stati precedentemente presentati gli svantaggi, una volta pesati questi con i punti di forza sulle quali si basa il DSL si capisce il perchè comunque è stato scelto questo approccio. Infatti considerando di avere a disposizione un modello chiaro e condiviso a livello aziendale, il vantaggio che si ottiene dal punto di vista temporale durante la fase di generazione del codice risulta essere sostanziale e permette inoltre la produzione di artefatti già dalle fase di analisi i quali svolgono il duplice scopo di modello di sistema utile alla comprensione dello stesso e parte fondante per la generazione di codice. Altro aspetto fondamentale, almeno nell'abito dell'IoT nel quale è inserito il nostro progetto è la forte riusabilità del codice garantita dall'uso del DSL, questo requisito, anche se identificato come non funzionale, risulta essere importante anche ai fini del corso e difficilmente con altre modalità operative risulta essere ottenibile.

7 Work plan

8 Project

8.1 Structure

8.2 Interaction

8.3 Behavior

9 Implementation

10 Testing

11 Deployment

12 Maintenance

See [?] until page 11 (CMM) and pages 96-105.

13 Information about the author

Beatrice Mezzapesa	Alessia Papini	Lorenzo Pontellini