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 piano di azioni;
– Definizione di azione temporizzata;
– Definizione di azione interrompibile;
– Definizione di console remota.

5.1 Piano di azioni

Un piano a questo livello può essere identificato da un nome e composto da una serie di azioni 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.

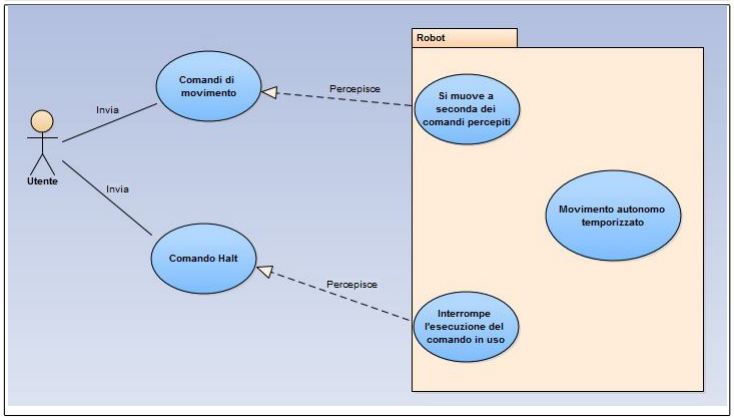
5.4 Console remota

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

6 Problem analysis

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 che vede come mezzo comunicativo la rete.

Gestione dei messaggi Ricopre un ruolo fondamentale la gestione dei messaggi i quali, come già 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.
Un'ulteriore suddivisione che si può attuare sulle specifiche dei messaggi è:

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

– REQUEST-RESPONSE: Si considera l'invio di un messaggio 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 messaggio e si aspetta un messaggio di risposta conforme a quanto richiesto, con l'assunzione della effettivo soddisfacimento della richiesta.

Per la realizzazione di azioni temporizzate è necessario poter impartire comandi che già possano esprimere la durata dell'azione da svolgere. E' indispensabile un meccanismo che permetta di valutare l'effettiva durata dell'azione mentre questa viene eseguita, non permettendo l'esecuzione del comando successivo, fino a quando l'azione in corso non sarà terminata. Quindi risulta essenziale determinare ed individuare il concetto di "termine" del comando in esecuzione. Per definire azioni interrompibili è necessario che il flusso di esecuzione di comandi possa essere bloccato e di conseguenza sarà indispensabile sia poter percepire il segnale inviato dalla console remota sia che il sistema rimanga reattivo ed in grado di fermare il piano in corso di svolgimento. Infine, per gestire completamente le azioni eseguite dal robot tramite console remota, il DDR dev'essere in grado di riconoscere ed eseguire i comandi percepiti.

CONCLUSIONI PARZIALI Il sistema in esame è in grado di darsi dei comandi e reagire in un certo modo nel momento in cui si verificano situazioni prestabilite. 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, in questo caso interrompedo il piano in esecuzione. Per queste motivazioni si introduce il concetto di piano definito come sequenza di comandi temporizzati.

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.

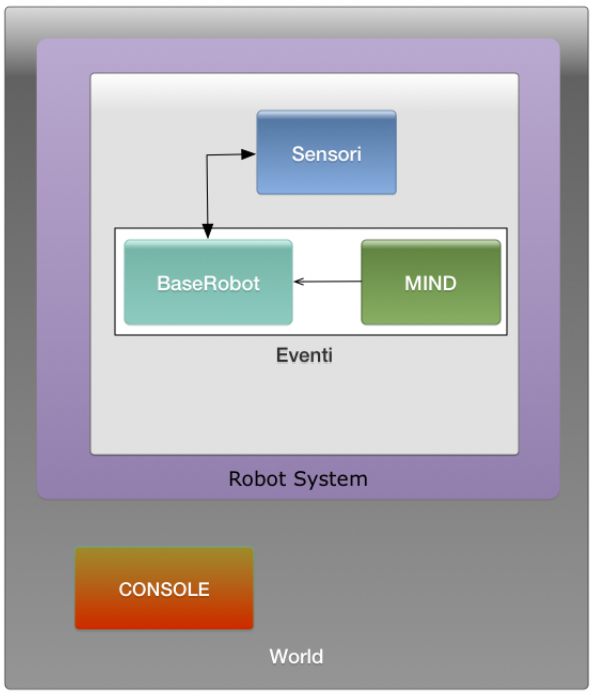
Struttura Da quanto detto, si identificano principalmente tre sottosistemi:

– 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. 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 un'infrastruttura ad eventi avendo identificato, in fase di analisi, alcune caratteristiche coerenti per gestire al meglio le informazioni provenienti dai sensori.

Struttura



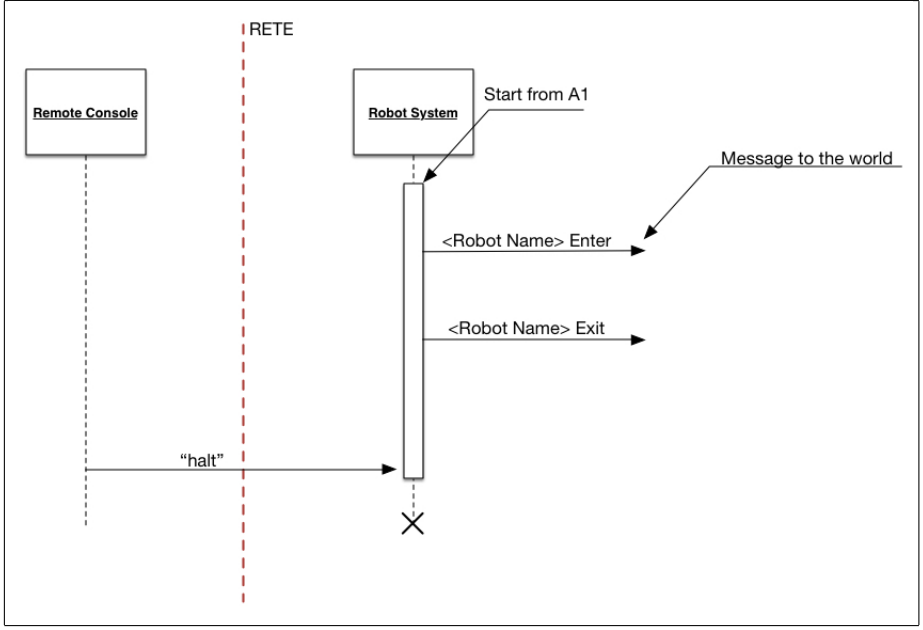
All'interno del diagramma si è voluto anche rappresentare il **"mondo"**, situando così il robot ed identificando il contesto di esecuzione del piano stesso. Un'ulteriore sottosistema identificato, come già detto, è la **console remota** che, ai fini dettati dal testing del sistema, verrà posta all'interno del sistema 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 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:

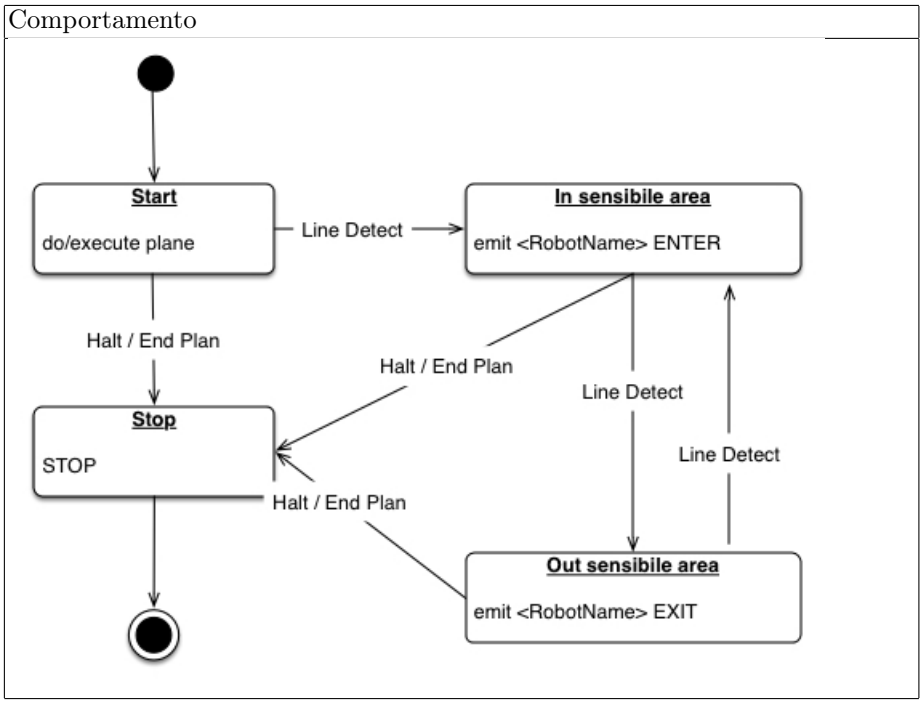
msg(MSGID, MSGTYPE, SENDER, RECEIVER, CONTENT, SEQNUM)

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.



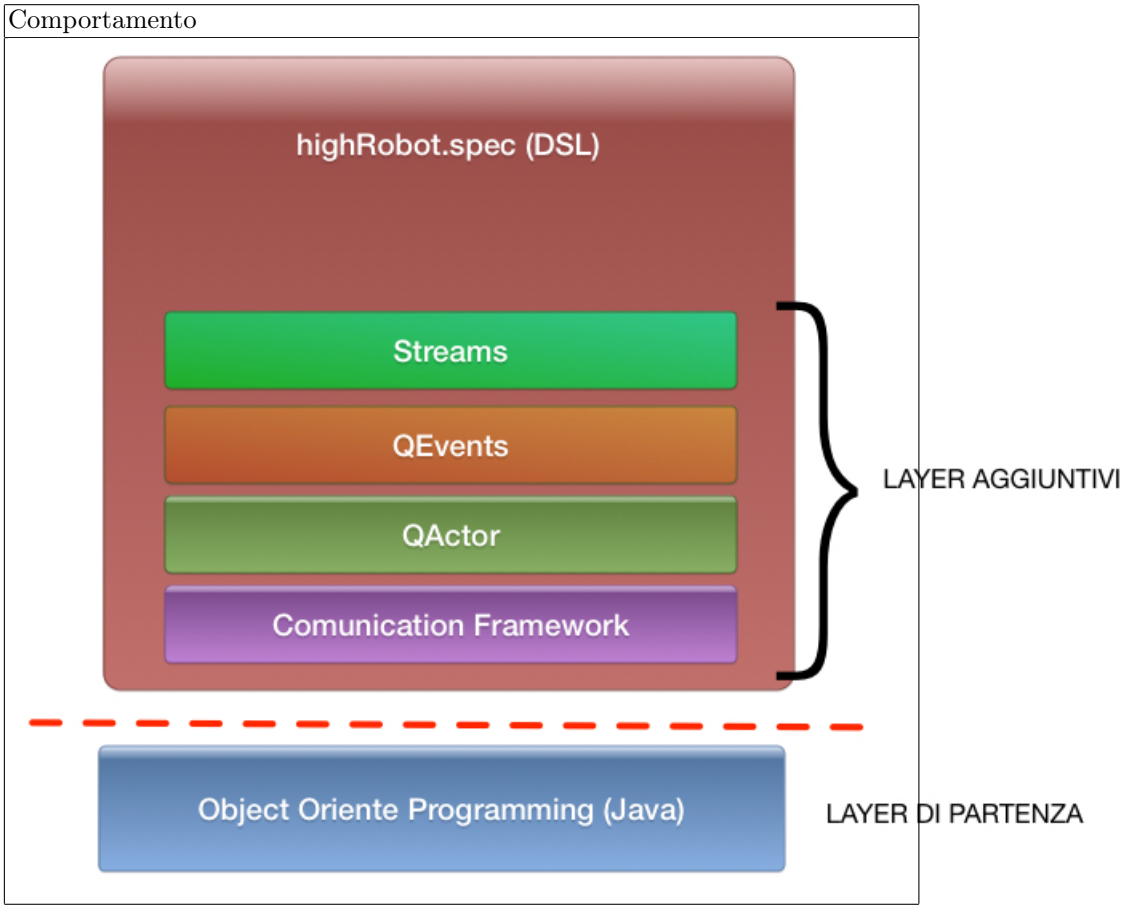
Per descrivere al meglio il comportamento del sistema abbiamo deciso di utilizzare un DSL (domain specific language), utile per descrivere la configurazione del robot e adatto alla prototipazione rapida. Il DSL è solo un punto di partenza, è abbastanza facile per la modifica della sintassi mentre la 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.

6.2 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 carat-

teristica di permettere la generazione di codice del tutto automatica e quindi l'abbattimento dei 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.



6.3 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 esperti 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 capire 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, nè in fase di utilizzo per la generazione di codice nè 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 stessa per modificare un comportamento, o di una modifica della semantica, presente all'interno di uno dei layer definiti, le ripercussioni che si avranno saranno visibili 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 maniera impredicibile su ciò che è già stato creato in precedenza.

Dalle problematiche identificate 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'ambito 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

13 Information about the author

Alessia Papini	Beatrice Mezzapesa	Lorenzo Pontellini