Ingegneria dei Sistemi Software
Approfondimento
Terza 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 Problem Analysis

Riproponiamo alcuni concetti relativi all'analisi del problema presentati nella precedente relazione[1], utili nella contestualizzazione di questo lavoro:

> *"Visto il contesto nel quale ci si pone, le problematiche identificate hanno portato, come citato all'interno dell'analisi dei requisiti, alla definizione 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, sfruttando 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. Da qui in avanti viene definito* ***piano*** *come sequenza di azioni.*
> *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 dell'azione ed eventualmente proseguire con la successiva.*

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

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 del 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 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 questa può essere eseguita senza che il chiamante debba attendere la fine dell'esecuzione dato che il controllo viene immediatamente restituito al chiamante.*

2 Abstraction Gap

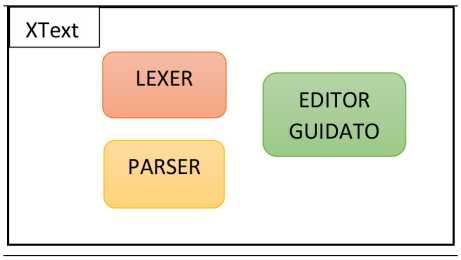
L'implementazione di quanto descritto precedentemente può essere realizzata utilizzando il linguaggio Java, seguendo questo approccio però non si è in grado di esprimere ad un sufficiente livello di astrazione le tematiche sopra citate e si nota come il dislivello dal punto di vista tecnologico sarebbe troppo ampio per essere colmato in un unico step computazionale, comportando inoltre un'elevato dispendio di tempo.
Per ovviare a questo, puntando a rendere più semplice la fase implementativa, si decide di sfruttare un'infrastruttura composta da una serie di framework quali: **QActor** e **QEvent** proposta nel corso di "Ingegneria dei Sistemi Software". Di seguito verrano proposti i concetti fondamentali alla base di tale infrastruttura utili per un'utilizzo ottimale della stessa.
QActors è un framework custom utile a mostrare come i progettisti affrontano l'analisi, la progettazione e l'implementazione di sistemi software utilizzando uno stile a message-passing piuttosto che un stile tradizionale basato sugli oggetti. Un QActor è un'entità attiva che ha lo scopo di eseguire le operazioni di base, quali la ricezione e l'invio di un messaggio, ma può anche eseguire un comportamento espresso da un piano, sequenza di azioni/attività che hanno un tempo finito per essere eseguite. Questo tipo di modello computazionale potrà essere considerato come assodato per le future implementazioni del problema. I QActors sono stati utilizzati anche nella prima fase di progetto, pur non essendo strettamente necessari in un sistema concentrato che non ha l'esigenza di considerare la comunicazione attraverso la rete. In particolare come base di partenza si è utilizzata l'entità RobotActor, che deriva da QActor, e permette l'interazione con il BaseRobot. Nella seconda fase invece, il contesto da prendere in considerazione è distribuito, in quanto è necessario interagire con il robot per interrompere un'azione, anche in questo caso sarà utilizzata l'entità RobotActor esteso con la

possibilità di ricevere comandi inviati dalla console remota oppure dal browser. Possiamo, quindi, considerare come primo modello per gestire la comunicazione quello a message-passing. In questo caso, però, eseguendo un'azione di una durata prefissata non saremo in grado di interromperla in alcun modo (in quanto questo modello non è proattivo/reattivo).
Prendendo in considerazione la possibilità di gestire le azioni eseguite dal robot come un automa a stati finiti e considerando il contesto distribuito nel quale ci troviamo, quindi la presenza della rete, per poter gestire i cambiamenti ai quali il robot deve reagire è necessario introdurre il concetto di Evento e più precisamente QEvent. QEvent è un framework che permette l'implementazione di sistemi software eternogenei in grado di adottare un modello event-driven. Un evento è un'informazione che viene emessa da una sorgente, che può essere esterna nel caso in cui la sorgente sia fisica come la console remota o interna nel caso in cui la sorgente sia un componente software come il robot.
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'secuzione 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. Ogni azione eseguita dal robot verrà, quindi, modellata come automa a stati finiti con tante transizioni in diversi stati, quanti saranno gli eventi che potranno essere ricevuti. Per poter realizzare un'azione che durante la sua esecuzione possa essere interrotta, si ha la necessità di un'entità distinta che si occupi di percepire e reagire gli eventuali eventi ricevuti, ovvero un **EventHandler**. A sua volta quest'entità avrà il compito di interrompere l'effettiva esecuzione dell'azione sul robot.

3 Future Work

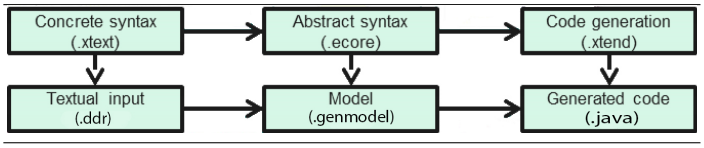
Dalle problematiche emerse in fase di analisi dell'applicazione proposta, e dalle soluzioni trovate, si nota come occorra un metodo per formalizzare in maniera adeguata i concetti appresi, così da poterli riutilizzare nel risolvere future problematiche che presentano caratteristiche simili a quelle affrontate.
Nasce quindi l'esigenza di definire una meta-modellazione di concetti che perme-

tta di costruire e sviluppare delle regole e delle teorie applicabili a problematiche simili. Il passo successivo prevede la definizione di un **linguaggio custom**, in questo caso **domain specific**, che consenta una produzione di codice automatizzata e che permetta di abbattere i costi realizzativi legati alle soluzioni identificate ed inoltre garantisca di passare automaticamente da una modellazione dichiarativa del funzionamento del sistema a codice eseguibile, favorendo una veloce prototipazione.
Avendo definito i concetti necessari all'interno dell'infrastruttura, della quale ci siamo avvalsi per riuscire a risolvere il problema proposto, si vogliono formalizzare questi concetti che rappresentano il meta-modello di partenza. Per la definizione di un linguaggio custom ci si avvale della tecnlogia **Xtext** la quale ci permette di abbattere i tempi realizzativi da diversi mesi a pochi giorni. Xtext è una tecnologia open source che permette di generare il **parser**, il **lexer**, un **editor guidato della grammatica** oltre ad altre fuzionalità.



Xtext

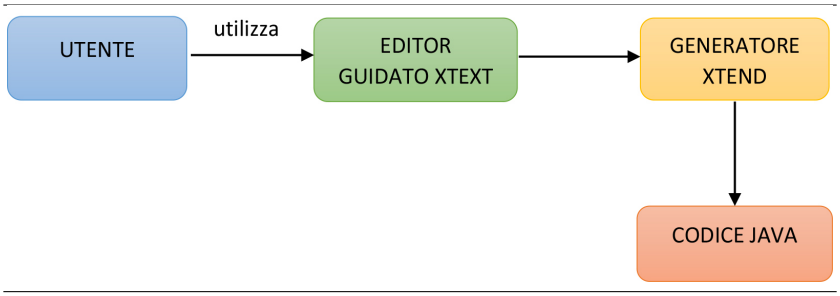
L'utilizzo della tecnologia del DSL risulta essere utile inoltre perchè fornisce un modello formale, secondo le tre dimensioni fondamentali di analisi, quali struttura, interazione e comportamento del sistema in esame, evitando le problematiche alle quale l'uso di UML non riesce a fare fronte.
Il linguaggio in cui è definito il meta-modello prende il nome di **Ecore** e i concetti che vengono mappati all'interno del linguaggio sono quelli contenuti nei QActor e nei QEvent precedentemente presentati.



Utilizzo del framework Xtext per l'implementazione del DSL

Realizzando un linguaggio di programmazione specifico per il contesto dei robot, è più opportuno scegliere un linguaggio che sia interpretato, in modo da non doverlo ricompilare ad ogni modifica effettuata. L'approccio garantito dall'utilizzo della tecnlogia xtext, permette di ottenere codice sia compilato che interpretato

infatti, nel DSL in questione è possibile sia generare codice Java che dev'essere compilato, sia utilizzare codice, come nel caso di plan.txt (nel quale viene specificata la serie di azioni che il robot deve eseguire), che dev'essere interpretato ed eseguito dal robot senza bisogno di essere ricompilato.



Utilizzo del framework Xtext dall'utente

4 Information about the author

Alessia Papini	Beatrice Mezzapesa	Lorenzo Pontellini