BoundaryWalk24

BW24-Rerquirements

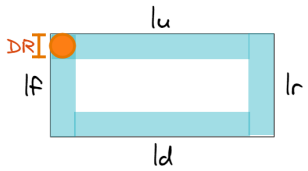
Costruire un sistema software che induce il *Il VirtualRobot*:

- *BW-Req1* : a percorrere (una volta) il bordo perimetrale (libero da ostacoli) della stanza rappresentata ne *La scena di WEnv*
- *BW-Req2* : a fermarsi di 5 sec quando rilevato da un Sonar della stanza
- *BW-Req3* : a fermarsi quando il sonar realizzato nel *Progetto sonarqak24* rileva un ostacolo.

BW24-Requirement analysis

Modello della stanza

La stanza è uno spazio piano Euclideo delimitato da bordi.



- Il *Bordo perimetrale* ha lunghezza **lf+ld+lr+lu**.
- Poichè la stanza è rettangolare, si ha **lf==lr && ld==lu**
- **DR=2R**, essendo **R** il *raggio del cerchio* in cui può essere racchiuso il *Il VirtualRobot*.

La locazione Home

Lo *Stato iniziale del virtualrobot* è l'angolo superiore sinistro, detto **Home**.

Sonar della stanza

Sono dispositivi che rilevano la distanza dal robot. Attualmente,un sonar è posto sul bordo **wallUp**. Quando rilevano il robot, i sonar emettono le seguenti informazioni:

```
{"sonarName": "<sonarName>", "distance": <int>, "axis": "AXIS" }
AXIS = x | y  //a seconda dell'orientamento del sonar

//Esempio:
{"sonarName":"sonar1","distance":-6,"axis":"y"}
```

Sonar esterno

Il sonar del *Progetto sonarqak24* è un dispositivo che emette l'*evento qak*:

```
Event obstacle  : obstacle(D)
```

rappresentato dalla stringa **msg(obstacle, event, sonar24, ANY, obstacle(DISTANCE), SEQNUM)**

BW24-Requirement architecture

obstacle come evento è al momento solo indicativo e non prescrittivo

Il modello *bw24res.qak* viene introdotto per indicare (in modo formale) che il sistema è composto da:

- il componente esterno **sonar24** che produce (in modo da precisare) informazioni riguardo alla rilevazione di un **obstacle**
- il componente esterno *VirtualRobot23* che rappresenta il robot da gestire e un ambiente che include i sonar della stanza che emettono stringhe della forma

```
{"sonarName": "<sonarName>","distance":<int>,"axis":"AXIS"}
```

- un componente applicativo denominato **bw24core**, che deve:
  - gestire le informazioni riguardo a **obstacle** (al momento rappresentate come eventi).
  - inviare al *VirtualRobot23* comandi di movimento in forma di stringhe *cril*
  - percepire i *messaggi di stato* emessi dal robot e dai sonar dell'ambiente *WEnv*.

BW24-Problem analysis

Il problema implica la costruzione di un sistema software che:

- presenta funzionamento proattivo, legato al movimento autonomo del robot lungo il bordo perimetrale della stanza;
- presenta funzionamento reattivo, quando deve reagire agli eventi dei sonar.

Inoltre osserviamo che:

- l'emissione di un *Messaggio di stato* da parte di *WEnv* avviene solo in relazione ad una mossa che il robot virtuale sta compiendo;
- l'emissione di un *Messaggio dall'ambiente* avviene anch'esso in relazione al movimento del robot virtuale e in modo anche ripetuto;
- in sistemi reali l'ambiente potrebbe essere sorgente di informazione non legata al movimento del robot, come ad esempio già accade con il sonar del *Progetto sonarqak24*.

Le principali problematiche poste dai requisiti sono riconducibili alle seguenti domande:

- **BW24-P1**: come può il componente **bw24core** reagire agli eventi mentre sta eseguendo il suo funzionamento proattivo?
- **BW24-P2**: come avviene l'invio al robot di comandi *cril* da parte di **bw24core**? Ricordiamo che:
  - il virtualrobot può essere comandato inviando stringhe *cril* in modo sia *sincrono* (via **HTTP**) sia *asincrono* (via **WS**)
- **BW24-P3**: come avviene la percezione delle informazioni emesse dai sonar da parte di **bw24core**? Ricordiamo che:
  - *WEnv* emette sulla WS un *Messaggio di stato* sulla distanza rilevata da un sonar;
  - **sonar24** costruisce informazione sulla rilevazione di un **obstacle**, che può essere resa disponibile in vari modi.

Ad una prima analisi sembra dunque opportuno:

- avvalersi di *Interaction* per creare una connessione HTTP o WS con *VirtualRobot23*, su cui inviare e ricevere informazioni. Questo obiettivo viene facilitato dal supporto *VrobotLLMoves24* che realizza una **connessione WS** in modo da poter non solo inviare comandi di movimento al robot, ma anche ricevere i *messaggi di stato*.
- avvalersi della *Qak infrastructure* per realizzare le interazioni con *sonar24*. Questo accade ovviamente in modo implicito impostando un modello qak del sistema.

Partendo da queste ipotesi, affermiamo che, riguardo alle tecnologie:

- **BW24-P1**: è posibile interrompere un movimento in corso con il comando **halt**.

- **BW24-P2**: i comandi possono essere inviati su una connessione WS, avvalendosi di un opprotuno supporto, quale ad esempio *VrobotLLMoves24*.
- **BW24-P3**: i dati dei sonar della stanza sono convertiti dal supporto *VrobotLLMoves24* in eventi, mentre si ritiene possibile intervenire in fase di progettazione sul *Progetto sonarqak24* per rendere disponibile l'informazione sulla rilevazione di un **obstacle** nel modo ritenuto più adeguato alla presente applicazione.

BW24 - Un approccio naive

Molto spesso si segue l'impulso di **saltare direttamente alla progettazione e alla codifica** a partire dalle tecnologie disponibili, senza soffermarsi troppo sulla analisi del problema. Ad esempio, una possibile implementazione, che interagisce direttamente con il *VirtualRobot23* potrebbe essere la seguente:

bw24naive.qak che:

- Accede al virtual robot usando *VrobotLLMoves24*.
- Invia un comando *forward* con tempo sufficientemente lungo per garantire il contatto con la parete di fronte.
- Una volta percepito il contatto, invia un comando *turnLeft* e prosegue con lo stesso schema altre tre volte.

Tuttavia, questo modo di procedere però non è conforme ai principi de *La Clean Architecture* e dell'*Dependency inversion*.

Riflettiamo sulle dipendenze

BW24 - Dipendenze legate alla interazione

Secondo i principi de *La Clean Architecture*, il livello applicativo (cioè **bw24core**) non dovrebbe dipendere dai vincoli d'uso imposti da *VirtualRobot23*.

Dipendenze dai linguaggi di comunicazione

Poichè *VirtualRobot23* non è un POJO, ma un componente con cui si interagisce mediante messaggi, le dipendenze sono ora rappresentate dal linguaggio con cui inivare comandi al robot e con cui percepire i *messaggi di stato* del robot e dei sonar.

Il linguaggio aril

Poichè l'uso di un robot virtuale è **solo un passo intermedio** verso un robot fisico, che potrebbe avere un linguaggio di comando diverso, può essere opportuno introdurre un linguaggio di comando 'technology-independent', che denominiamo **linguaggio aril** (**Abstract Robot Interaction Lanaguage**) con cui esprimere i comandi-base di spostamento con la sintassi che segue:

```
MOVE = w | s | l | r | h | p

Dispatch cmd : cmd( MOVE ) "aril MOVE=w | s | l | r | h | p"
```

Osservabilità

Dualmente, si ritiene opportuno mappare i *messaggi di stato* in informazione osservabile da componenti esterni

Verso un basicrobot logico

Le considerazioni precedenti prefigurano l'opportunaità di introdurre un componente (un actor) che possa fare da 'adapter' tra l'applicazione e i dettagli tecnologici sull'uso di uno specifico robot.

BW24-Logical architecture

Visto l'insieme delle problematiche, l'architettura logica può specificare in modo preciso l'insieme iniziale dei componenti, senza indicare in modo preciso le mosalità di interazione, che devono essere analizzate, discusse e decise in fase di progettazione.

Il modello *bw24analisi0.qak* è solo indicativo delle informazioni (in forma di eventi) che i diversi componenti devono generare e percepire.

```
Dispatch cmd    : cmd( MOVE ) "aril MOVE=w | s | l | r | h"
Event vrinfo    : vrinfo(A,B)  "info emessa da vrrobot"
Event sonardata : sonar(D)     "emesso da WEnv"
Event obstacle  : obstacle(D)  "emesso da sonar24"
```

I componenti sono rappresentati come attori in unico contesto, dando al nome il suffisso **mock** per indicare

che si tratta di un modello provvisorio.

Gli eventi **vrinfo** e **sonardata** derivano da quanto riportato in *VrobotLLMoves24*.

BW24-Test Plans

In assenza di indicazioni precise sulla interazioni, la loro specifica formale deve essere rimandata alla fase progettazione.

BW24-Project

BW24-Testing

BW24-Deployment