VrobotLLMoves24

progetto virtualrobotusage

Vogliamo realizzare le interazioni tra un livello applicativo modellato com un *actor* supportato dalla *Qak infrastructure* e il *VirtualRobot23*.

Scopo del componente

L'obiettivo è reralizzare un POJO (*VrobotLLMoves24* anche detto **VRHL24**) che funga da supporto per un *actor qak*, detto **owner**, che:

1. fornisce metodi con cui **owner** può inviare *Comandi di movimento* al *VirtualRobot23*;
2. *trasforma in eventi* le informazioni emesse su *WebSocket* (**WS**) da *WEnv*.

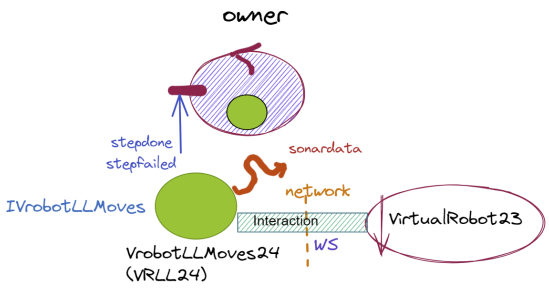
- Per realizzare lo scopo, selezionamo il modo di *Interazione asincrona* con il *VirtualRobot23*.
- Così facendo, sarà possibile inviare comandi in modo *fire-and-forget*, senza precludere la possibilità di ottenere informazioni da WEnv (un *Messaggio di stato* inviato da **WEnv** a tutti i client connessi) anche prima della terminazione dell'operazione.

Il progetto *unibo.basicomm23* definisce la classe *WsConnection*, che offre all'application designer strumenti e meccanismi utili a gestire questa situazione.

VRHL24-Informazioni da WEnv come eventi

Il POJO *VrobotLLMoves24*:

- apre una connessione (**wsconn**) su WS con *VirtualRobot23*
- opera come observer su **wsconn**
- trasforma le informazioni ricevute su **wsconn** in eventi:




```
sonardata : sonar(D)
vrinfo    : vrinfo(X,Y)
```

Per operare come *observer*, il supporto estende *ApplAbstractObserver* e si registra come osservatore sulla *WsConnection*.

```
public class VrobotLLMoves24 extends ApplAbstractObserver implements IVrobotLLMoves{
```

Un dettaglio sulla tecnologia

Attenzione ai Thread

Il metodo update che *VrobotLLMoves24* deve definire in quanto observer, viene eseguito nel Thread (di nome Grizzly) della libreria di supporto alle WS **tyrus-standalone-client**, che promuove l'uso del metodo annotato onmessage per la elaborazione di un *Messaggio di stato* inviato sulla WS.

In altre parole, la libreria non offre metodi bloccanti per la ricezione di messaggi, ma invoca direttamente metodi di callback del livello applicativo.

Azioni di callback

Le azioni applicative che operano come *funzioni di callback* possono essere eseguite:

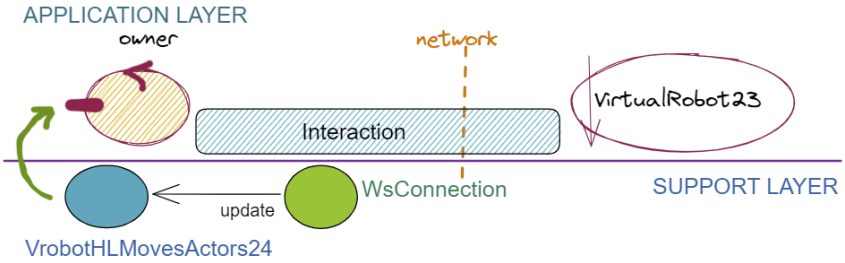
1. dal supporto *VrobotLLMoves24*, in modo simile a quanto fanno framework come *Node.js* ->;
2. dall'attore **owner**, una volta che il supporto gli abbia notificato la ricezione dei *messaggi-statocomando*.

Relazione supporto-applicazione

La esecuzione di una azione di callback da parte del supporto risulta problematica in quanto il metodo update viene eseguito nel Thread Grizzly della libreria. Per non perdere le informazioni applicative di contesto, è opportuno (in pratica indispensabile) che:

le callback siano eseguite nel Thread applicativo

La parte di supporto che gira in Grizzly, può essere visto come **produttore** di informazione che può essere inserita nella coda di ingresso dell'owner, in modo che possa essere **consumata** dalll'applicazione nel suo proprio Thread.



VRHL24-Inizializzazione

L'inizializzazione del supporto crea la connessione su WS, si registra come osservatore e definisce la struttura del messaggio (**toApplMsg**) da inviare all'applicazione, nel caso in cui l'applicazione sia un attore o meno.

```
public class VrobotLLMoves24 extends ApplAbstractObserver implements  IVrobotLLMoves{
  protected String vitualRobotIp = "localhost";
  protected ActorBasic owner;
  private Interaction conn;
  protected String toApplMsg;                 //message to the application
  protected String asynchMoveResult = null;   //for observer part


  //Factory method
  public static VrobotLLMoves24 create( String vitualRobotIp, ActorBasic owner ) {
      return new VrobotLLMoves24( vitualRobotIp, owner );
  }
  //Constructor
  public VrobotLLMoves24(String vitualRobotIp, ActorBasic owner) {
      connect(vitualRobotIp, owner);
  }
  protected void connect(String vitualRobotIp, ActorBasic owner) {
      this.vitualRobotIp = vitualRobotIp;
      this.owner         = owner;
      this.conn = ConnectionFactory.createClientSupport(
                      ProtocolType.ws,vitualRobotIp+":8091","");
      ((WsConnection) conn).addObserver(this);  //DIVENTA OSSERVATORE
      if( owner != null )
              toApplMsg = "msg(wenvinfo, dispatch, support, RECEIVER, CONTENT, 0)"
             .replace("RECEIVER",owner.getName());
      else
          toApplMsg = "msg(wenvinfo, dispatch, support, RECEIVER, CONTENT, 0)"
          .replace("RECEIVER","alien");
  }
```

VRHL24-Comandi di movimento

I comandi di movimento messi a disposizione dal supporto sono rappresentati dai metodi della seguente interfaccia, ispirata ai *Comandi-base per il robot in cril*:

IVrobotLLMoves

```
public interface IVrobotLLMoves {
    //Ispirate da VirtualRobot23
    public void turnLeft() throws Exception;
    public void turnRight() throws Exception;
    public void forward( int time ) throws Exception;
    public void backward( int time ) throws Exception;
    public void halt() throws Exception;

    //Nuove operazioni
    public boolean step(long time) throws Exception;
}
```

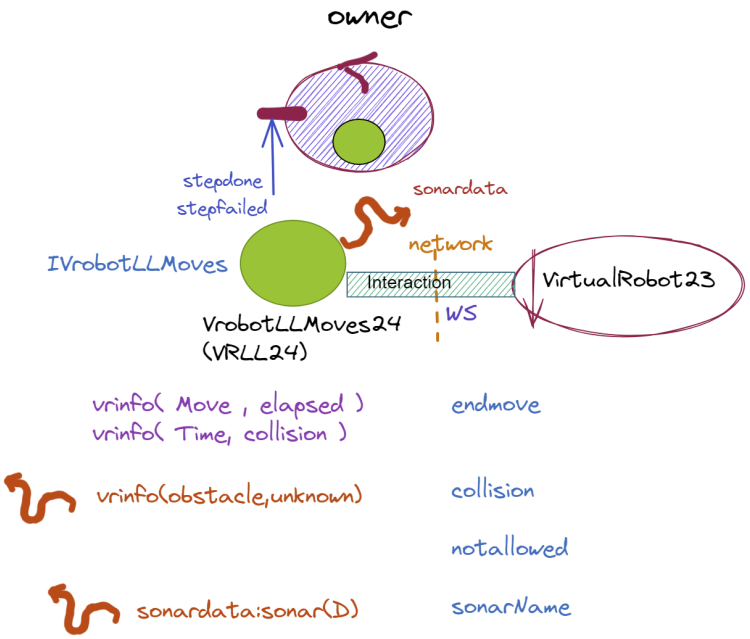
Le *mosse hanno tutte una durata limitata*, ma le interazioni sulla **ws** sono di tipo asincrono e per sapere quando una mossa è terminata occorre gestire il *Messaggio di stato* inviato sulla **ws** da WEnv.

messaggi-statocomando

La forma dei messaggi di stato relativa alla terminazione del comando è:

```
{"endmove":"true", "move":"moveForward"}            //SUCESSO
{"endmove":"false","move":"moveForward-collision"}  //FALLIMENTO
```

D'ora in avanti, denomineremo questi messaggi come messaggi-statocomando.



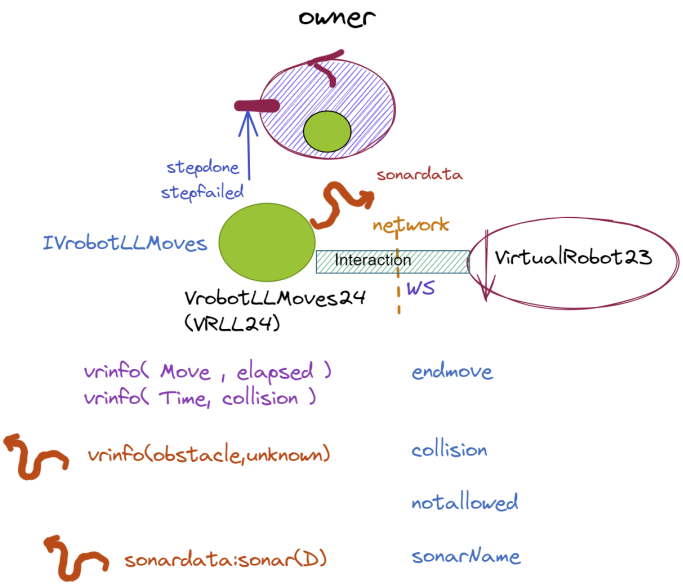
Notiamo che, rispetto ai *Comandi-base per il robot in cril*, in *IVrobotLLMoves* è definita la nuova operazione *step*.

step

- muove in avanti il robot per un tempo dato, restituendo il *boolean* **true** se il movimento termina con successo e **false** nel caso il movimento non possa essere completato

il metodo *step* esegue una richiesta bloccante, pur avendo inviato sulla WS un messaggio in modo asincrono.

Chi è l'owner?



owner: un componente applicativo

OPPURE

un servizio (actor vrqak) che offre:

Request step : step(TIME) "ASINCRONO"
Reply stepdone : stepdone(V) for step
Reply stepfailed : stepfailed(DURATION, CAUSE) for step

Request cmd : cmd(MOVE,T) "MOVE = w|s|a|d|p "
Reply cmddone : cmddone(R) for cmd
Reply cmdfailed : cmdfailed(T,CAUSE) for cmd

Si veda *vrqak*.