Software Engineering
Explore and Navigate
(titolo provvisorio)

Dario Pasquali

Alma Mater Studiorum – University of Bologna
viale Risorgimento 2, 40136 Bologna, Italy
dario.pasquali@studio.unibo.it

1 Introduction

Report dell'attività progettuale di Ingegeria dei sistemi software M.

2 Vision

3 Goals

4 The Problem

Si vuole realizzare un sistema in grado di mappare e navigare all'interno di un ambiente del quale non si possiede alcuna infromazione.
Il sistema deve permettere di:

- Esplorare l'ambiente tramite un Robot, generando una mappa digitale il più possibile fedele alla realtà;
- Muovere il Robot all'interno dell'ambiente, in modo che scelga in maniera autonoma il percorso più corto.

Si progetti il Robot individuandone le caratteristiche tecniche necessarie e l'hardware più adatto a risolvere il problema.
Si vuole inoltre dotare il sistema di una unità di controllo centrale chiamata Console, essa dovrà:

- Fungere da interfaccia di controllo, permettendo all'utente impartire comandi al Robot;
- Mantenere e visualizzare i dati relativi alla mappa, parziale o totale. (Per una futura espansione del progetto con N robot in coordinazione)

4.1 Note sull'Esplorazione

Il Sistema deve essere in grado di gestire due tipi di ambientazione:

- Ambiente CHIUSO: come abitazioni, cortili, etc... . L'esplorazione deve terminare quando tutta l'area è stata mappata, quando viene comandato in maniera esplicità dall'utente, o dopo un tempo massimo configurabile;
- Ambiente APERTO: qualsiasi spazio aperto senza delimitazioni fisiche. L'esplorazione deve terminare quando esplicitamente comandato o dopo un tempo massimo configurabile.

Durante l'esplorazione la mappa deve essere visualizzata sulla Console ed aggiornata a runtime, mano a mano che viene rilevata.

4.2 Note sulla Navigazione

Per quanto riguarda la Navigazione, l'utente deve poter specificare il punto di partenza (Start) e il punto di arrivo (Goal), in questo caso è sua premura collocare il Robot nel punto Start indicato.

In alternativa può specificare unicamente il punto Goal, il Robot considererà la sua attuale posizione come punto di Start.

In ogni caso esso dovrà scegliere in maniera autonoma il percorso più breve per navigare da Start a Goal. Se, durante la navigazione, rileva un ostacolo non previsto, il Robot deve considerarlo come dinamico, fermarsi ed attendere un tempo predefinito. Se al termine di questa attesa vi è ancora l'ostacolo, il Robot dovrà elaborare un percorso alternativo per raggiungere il Goal.

L'utente deve poter interrompere la Navigazione in qualsiasi momento.

4.3 Note Tecniche

Il Sistema deve permettere il controllo remoto del Robot da parte della Console.

Il Robot dovrà essere realizzato sfruttando come punto di partenza la scheda integrata Raspberry Pi. Nella progettazione si mantenga un adeguato equilibrio tra costo e prestazioni, selezionando i soli componenti utili alla soluzione del problema.

La Console dovrà essere il più possibile indipendente dalla macchina sulla quale è installata, per facilitare il più possibile la distribuzione. Si richiede una versione Desktop, su sistema Windows, e una mobile su sistema Android.

5 Requirement analysis

Dopo una sessione di confronto più accurato con il committente, questo è ciò che è emerso analizzando i requisiti più a fondo.

Il sistema è composto da 2 entità:

- **Robot**: un agente autonomo e intelligente da realizzare con tecnologia raspberry e dotato di sensori e attuatori.
  Deve essere in grado di Esplorare l'ambiente, rilevando gli ostacoli, e i limiti fisici, creando mano a mano una mappa il più possibile accurata e fedele alla realtà.
  Questa mappa deve essere trasmessa a runtime alla seconda entità.

  Inoltre deve essere in grado, data una mappa dell'ambiente e due punti (Start e Goal), di navigare dal primo al secondo scegliendo in maniera autonoma e reattiva il percorso più corto.
- **Console**: una interfaccia software, realizzata su sistema Windos e Android, con cui l'utente può monitorare e controllare il comportamento del robot.

  Tramite la Console, l'utente deve poter gestire le due fasi principali della elaborazione:

  - Esplorazione: creazione della mappa dell'ambiente.
    Parametri: Tipo di Ambiente, Tempo Massimo.
  - Navigazione: movimento dal punto Start al punto Goal seguendo il percorso più breve.
    Parametri: punto Goal, punto Start (opzionale)

  Le due fasi devono essere interrompibili a piacimento. La Console deve mostrare inoltre la mappa aggiornata in tempo reale.

Il sistema deve essere in grado di gestire due tipi di ambiente:

- APERTO: si intende con ambiente aperto un qualsiasi ambiente dove non sono presenti elementi fisici che fanno confine massimo per l'esplorazione del Robot (ad esempio una strada). Inoltre in un ambiente di questo tipo è possibile utilizzare sensori come il GPS;

- CHIUSO: si intende con ambiente chiuso un qualsisi ambiente dove sono presenti dei confini fisici che limitano l'area esplorabile del robot (ad esempio una stanza). Per mantenere il sistema il più generale possibile, si suppone che in ambienti come questo non funzionino meccanismi di localizzazione satellitare.

La distinzione degli ambienti sarà fondamentale in fase di progettazione per definire il comportamento del robot in fase di Esplorazione.

La mappa, prodotta in fase di Esplorazione, deve inoltre essere memorizzata a lato Console.
Essa deve poter anche essere caricata nel sistema in un secondo momento, ed utilizzata per la Navigazione, senza dover ripetere il processo di Esplorazione.

5.1 Use cases

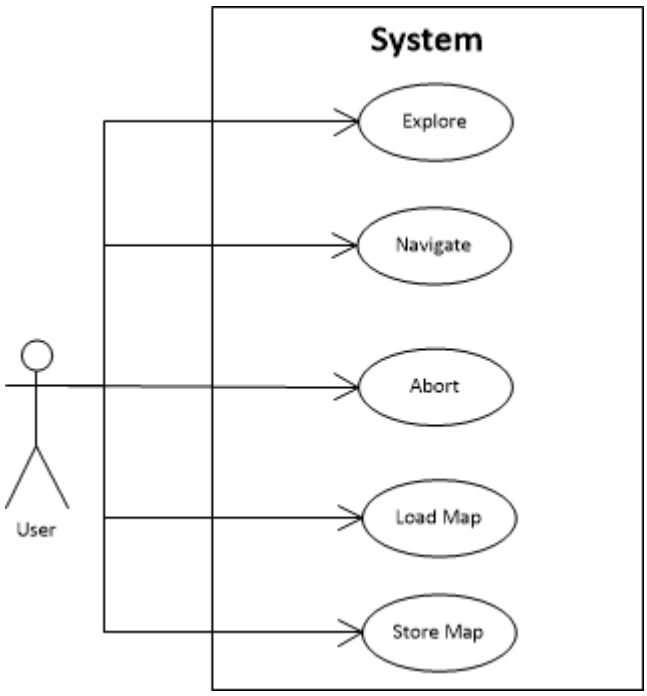

Fig. 1.

5.2 Scenarios

...

5.3 (Domain)model

CONSOLE MODEL

```
1
public interface IConsole {
3
public IMap explore(String EnvType, long maxMilsTime);  //modifier , primitive
5 public void navigate(double goalX, double goalY);       //modifier , primitive
public void navigate(double startX, double startY, double goalX, double goalY);
7 public void adbort();                                   //modifier , primitive

9 public void storeMap(IMap map, String filepath);        //modifier , primitive
public IMap loadMap(String filepath);                     //modifier , primitive
```

```
11
   public IMap getMap();                                    //property , primitive
13 public void getName();                                   //property , primitive

15 public String getDefaultRep();  // mapping , non-primitive
   }
```

— — — — — —

ROBOT MODEL

```
1  public interface IRobot {

3  public void move(String dir, double speed, double mils);//modifier , priitive
   public void move(String dir, int steps);            //modifier , priitive
5  public void turn(double angle, double speed);       //modifier , priitive
   public void stop();                                 //modifier , priitive
7
   public void setMap(IMap map);                       // property , primitive
9  public IMap getMap();                               // property , primitive

11 public void navigateTo(double goalX, double goalY); //modifier , priitive
   public void navigate(double startX, double startY, double goalX, double goalY);
13
   public IMap explore(String EnvType, long maxMilsTime);//modifier , priitive
15
   public String getName();                            // property , primitive
17 public String getDefaultRep();                      // mapping , non-primitive

19 }
```

5.4 Test plan

...

6 Problem analysis

Logic architecture

Per soddisfare i requisiti del problema non è sufficiente utilizzare dei semplici componenti POJO, è necessario invece dotarsi di una architettura eterogenea e distribuita.

Modello di Interazione

Per quanto riguarda l'interazione, ho scelto un modello fortemente Event Driven. Gli eventi vengono emessi sia dalla Console che dal Robot, la netta distinzione in fasi di eleaborazione permette di sapere con certezza quali eventi possono essere generati in un determinato momento.

Eventi e Contenuto Informativo

Durante il normale funzionamento del sistema si verificano le seguenti interazioni:

- **explore : explore(TYPE, MILS)** viene generato dalla Console quando l'utente decide di inziare l'Esplorazione dell'ambiente. Il payload indica il tipo di ambiente da esplorare (open / close) e il tempo massimo espresso in millisecondi;

- **navigate : navigate(GOAL)** oppure **navigate : navigate(START, GOAL)**, generato dalla Console quando l'utente decide di iniziare la navigazione, partendo da un mappa appena creata, oppure salvata precedentemente. Il payload indica la posizione GOAL e quella di START (opzionale) nel formato **position(X , Y)**;

- **abort : abort**, generato dalla Console, indica che l'utente vuole terminare l'elaborazione in corso, che sia una Esplorazione oppure una Navigazione. Non sono necessari dati aggiuntivi;

- **end : end**, generato dal Robot, indica che ha terminato l'esecuzione dell'ultimo comando, che sia di Esplorazione o Navigazione. Non sono necessari dati aggiuntivi;

- **update : update(ELEMENT)**, generato dal Robot, comunica che è stata esplorata una nuova posizione della mappa che, di conseguenza, deve essere aggiornata. ELEMENT è uno dei seguenti:
  - **obstacle(X, Y)**;
  - **clear(X , Y)**;
  - **border(X , Y)**;

Comportamento delle Entità

CONSOLE

```
RobotSystem exploreAndGo

/*
* ---------------------------------------------------
* EVENTS emitted by the system components
* ---------------------------------------------------
```

```
*/
// C to R, MILS is the maximum millisecond time
Event explore : explore(TYPE, MILS)

// C to R
Event navigate : navigate(START, GOAL)

// C to R
Event navigate : navigate(GOAL)

//C to R, abort exploration or navigation
Event abort : abort

//R to C, termination of exploration or navigation
Event end : end

//R to C, update the Map
Event update : update(ELEMENT)

/*
* EVENTS FROM THE GUI
*/
Event local_gui_command : command(C)


/*
* ------------------------------------------------------
* CONTEXT for the Robot as independent device (standdalone)
* ------------------------------------------------------
*/
Context ctxRobot ip [ host="localhost" port=8020] -standalone

/*
* ------------------------------------------------------
* CONTEXT for the Console
* ------------------------------------------------------
*/
Context ctxConsole ip [ host="localhost" port=8010]


/*
* ------------------------------------------------------
* CONSOLE with custom GUI
* ------------------------------------------------------
*/
```

```
54 QActor console context ctxConsole -g cyan{

56

58 Plan init normal
   println(console(start));
60 switchToPlan waitCommand

62 //-----------------------------------------------------------------------

64 Plan waitCommand

66 println("Wait for user command");

68 sense time(999999999) local_gui_command -> continue;
   [ ?? tout(X,Y) ] switchToPlan handleTimeout;
70 memoCurrentEvent;
   printCurrentEvent;
72
   onEvent local_gui_command : command(explore(TYPE, MILS))
74                  -> switchToPlan exploration;

76 [ !? have_map ] onEvent local_gui_command : command(navigate(START, GOAL))
                    -> switchToPlan navigation;
78
   [ !? have_map ] onEvent local_gui_command : command(navigate(GOAL))
80                  -> switchToPlan navigation;

82 onEvent local_gui_command : command(load(PATH))
                    -> switchToPlan loadMap;
84
   repeatPlan 0
86
   //-----------------------------------------------------------------------
88
   Plan exploration
90
   println("EXPLORE");
92
   // discard all old map data
94
   [?? msg(local_gui_command, "event", SENDER,
96                  none, command(explore(TYPE, MILS)), MSGNUM)]
                    emit explore : explore(TYPE, MILS);
```

```
98
   sense time(999999999) update, local_gui_command, end
100                -> continue, continue, continue;

102 [ ?? tout(X,Y) ] switchToPlan handleTimeout;
   memoCurrentEvent;
104 printCurrentEvent;

106 onEvent update : update(ELEMENT)
                   -> switchToPlan updateMap;
108
   onEvent local_gui_command : command(abort)
110                -> switchToPlan abortLastCommand;

112 onEvent end : end
                   -> switchToPlan endOfExploration;
114
   repeatPlan 0
116
   //-------------------------------------------------------------------------
118
   Plan endOfExploration
120 println("do you want to save the map or explore again??");

122 sense time(999999999) local_gui_command -> continue;
   [ ?? tout(X,Y) ] switchToPlan handleTimeout;
124 memoCurrentEvent;
   printCurrentEvent;
126
   onEvent local_gui_command : command(store(PATH))
128                -> switchToPlan storeMap;

130 onEvent local_gui_command : command(explore(TYPE, MILS))
                   -> switchToPlan exploration
132
   //-------------------------------------------------------------------------
134
   Plan updateMap resumeLastPlan
136 [?? msg(update, "event", SENDER, none, update(ELEMENT), MSGNUM)] println(update(

138 //-------------------------------------------------------------------------

140 Plan navigation resumeLastPlan
   println("NAVIGATE");
142
```

```
//-------------------------------------------------------------------------

[?? msg(local_gui_command, "event", SENDER, none,
                command(navigate(START, GOAL)), MSGNUM)]
                emit navigate : navigate(START, GOAL);

[?? msg(local_gui_command, "event", SENDER, none,
                command(navigate(GOAL)), MSGNUM)]
                emit navigate : navigate(GOAL);

sense time(999999999) local_gui_command, end
                -> continue, continue;

[ ?? tout(X,Y) ] switchToPlan handleTimeout;
memoCurrentEvent;
printCurrentEvent;

onEvent local_gui_command : command(abort)
                -> switchToPlan abortLastCommand;

onEvent end : end
                -> switchToPlan endOfNavigation

//-------------------------------------------------------------------------

Plan endOfNavigation
println("Robot reached the GOAL position");
switchToPlan waitCommand

//-------------------------------------------------------------------------

Plan loadMap resumeLastPlan
println("Map loaded");
[?? msg(local_gui_command, "event", SENDER, none,
                command(load(PATH)), MSGNUM)]
                addRule have_map

// <-- waitCommand

//-------------------------------------------------------------------------

Plan storeMap
println("Map saved");
[?? msg(local_gui_command, "event", SENDER, none,
                command(store(PATH)), MSGNUM)]
```

```
188                     addRule have_map;

190 switchToPlan waitCommand

192 //-------------------------------------------------------------------------

194 Plan abortLastCommand
    println("Abort");
196 emit abort : abort;
    switchToPlan waitCommand
198


200
    Plan handleTimeout
202 println("timeout!! GOODBYE")

204 }

206
    Robot scout QActor robot context ctxRobot{
208 Plan init normal
    println( "NEVER HERE. I'm just a place holder" )
210 }
```

è molto chiara la distinzione in fasi di elaborazione, è infatti possibile rappresentare il comportamento della Console, e dell'intero sistema come una macchina a stati finiti.

Le fasi principali sono Exploration e Navigation, troviamo inoltre le fasi di LoadMap, StoreMap e UpdateMap. Infine sono presenti numerose fasi che aiutano a rendere più semplice e funzionale l'elaborazione.

6.1 Abstraction gap

...

6.2 Risk analysis

...

7 Work plan

- ✓ Definizione dei requisiti
- ✓ Analisi del Problema
- ✓ modello per la Console (linguaggio naturale)
- ✓ modello per il Robot (linguaggio naturale)

- ✓ modello per la Console in maniera formale
- – modello per il Robot in maniera formale
- – Esplorazione "close" con confini definiti
- – Ricerca dei confini "close"
- – Esplorazione "open"
- – Navigazione

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

Photo of the author

References