Università degli studi di Modena e Reggio Emilia
Dipartimento di Ingegneria “Enzo Ferrari”
Corso di Laurea Magistrale in Ingegneria Informatica

Modellazione e prototipazione di una centralina per il rilevamento di fuoriuscita di gas naturale, gestito da agenti autonomi Java interfacciati su uno spazio di tuple

PROGETTO DI CORSO

Fabrizio Di Blasi | Sistemi distribuiti e intelligenza artificiale distribuita | a.a 2019/2020

Sommario

Introduzione

a. Finalità del progetto

In questo progetto si pone l'obiettivo di analizzare le possibili fuoriuscite di gas ed azionare dei sistemi di sicurezza in una condotta contenente GPL.
Il modellino presentato è provvisto di tre paratie, una comandata da un servomotore, le restanti due simulate tramite due coppie di led, rossi e verdi.
Il sistema è controllato da due microcontrollori, uno ha il compito di prelevare i dati, le misure, dai sensori di gas posti al di sopra della condotta, mentre l'altro, ha lo scopo di andare ad azionare le paratie in base ai comandi provenienti dal programma Java che si interfaccerà con questi due microcontrollori tramite protocollo seriale.
Il programma Java è il fulcro del progetto presentato, infatti, tramite uno spazio di tuple, mette in comunicazione 3 agenti software, modellati con dei thread autonomi. Due hanno il compito di andare a prelevare e fornire i comandi ai microcontrollori, il restante, invece, amministra lo spazio delle tuple ed in base a ciò che legge va ad inserire delle tuple che andranno ad attivare il microcontrollore che governa gli attuatori.

b. Componentistica adottata

Per la realizzazione del prototipo sono stati adottati i seguenti componenti:

Board Arduino Uno R3:

- Core
  - * ATmega328P @16Mhz
- Memoria
  - * Flash memory 32 KB
  - * SRAM 2 KB
  - * EEPROM 1 KB
- Periferiche
  - * Digital I/O Pins 14 (di cui 6 con PWM)
  - * PWM Digital I/O Pins 6
  - * Analog Input Pins 6

Servomotore MG996R:

Caratteristiche:
- * Peso: 55g
- * Dimensioni: 40.7×19.7×42.9mm
- * Stall torque: 10.5kg/cm (4.8v); 13kg/cm (6v)
- * Operating speed: 0.20sec/60° (4.8v);0.17sec/60°(6.0v)
- * Alimentazione: 4.8-6.6v
- * Temperatura: 0- 55°

Sensore di Gas MQ5:

Caratteristiche:

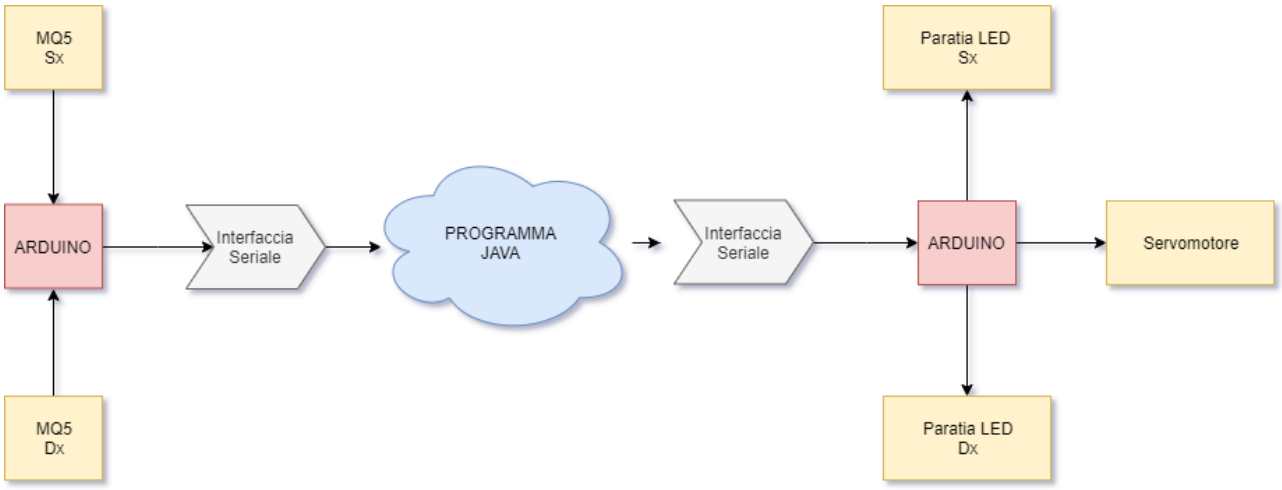
* Alimentazione a 5V DC
* 150mA di assorbimento
* Uscita TTL (D0) e analogica (A0)

La misurazione di questo sensore è affidabile solamente se il sistema è ad una temperatura posta tra -20 °C e +70 °C ed al di sotto del 95% di umidità.
Inoltre, un parametro da non sottovalutare è la concentrazione di ossigeno nell'ambiente, che in condizioni standard è del 21%. Un tasso maggiore o inferiore può causare degli errori di misura della concentrazione di gas.
Il sensore comunica col microcontrollore tramite un'uscita analogica e digitale (TTL).
L'uscita digitale assume valore logico

1 – se non c'è fuga di gas
0 – se viene rilevata la fuga di gas

Tale segnale è utilizzato per la gestione dell'interrupt, l'uscita analogica invece è quella che fornisce la misurazione in ppm della concentrazione di gas. Tale output viene fornito al convertitore ADC del microcontrollore.

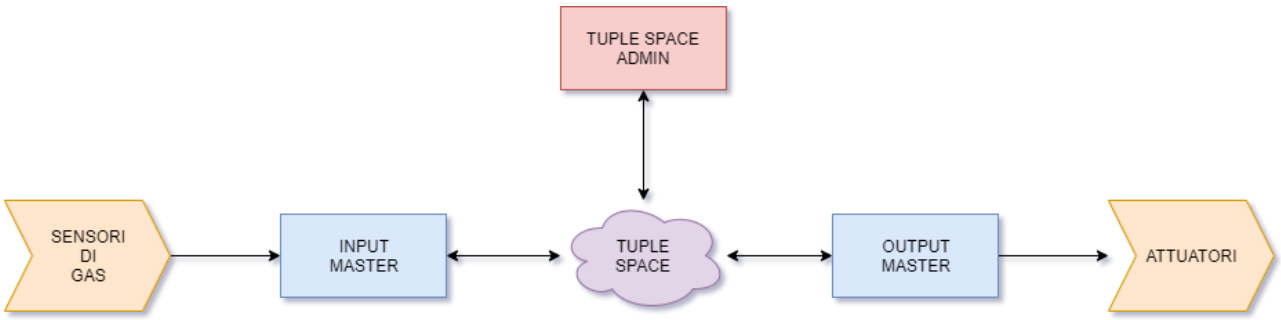
c. Schema a blocchi complessivo



Dallo schema a blocchi sopra, è possibile capire l'architettura del progetto e come i componenti sono interfacciati tra loro.
Per prima cosa si può vedere che il programma JAVA "dialoga" con i due microcontrollori con il protocollo seriale, dal verso delle frecce è possibile inoltre capire come avviene la comunicazione.
Il microcontrollore che ha il compito di misurare la presenza, o meno, di gas effettua una comunicazione monodirezionale con il sistema di controllo, il quale, in base all'andamento dei valori delle misurazioni in ingresso, sceglierà quale comando fornire al microcontrollore che azionerà gli attuatori.

d. Struttura del programma Java

Il programma Java è stato concettualizzato con tre agenti software molto semplici e modellati con delle classi.
Infatti, essendo la loro strategia di azione molto lineare e poco complessa, si è scelto di non usare i sistemi più evoluti e complessi come JADE e di preferire la stesura e delineazione della loro strategia direttamente in una classe java.



Nel diagramma sopra riportato si può vedere come si interfacciano tra loro questi tre agenti chiamati:

- Tuple Space Admin
- Input Master
- Output Master

Il primo agente *tuple space admin* ha il compito di analizzare le tuple che vengono inserite dall'agente *input master*. Infatti, il primo agente menzionato ha il compito di andare a controllare se i valori delle misurazioni di GAS sono al di sotto della soglia o meno.

Tali misurazioni sono pacchettizzate sottoforma di tupla dall'*input master*, infatti, esso preleverà i valori delle misurazioni provenienti dal sensore di destra e di sinistra e di conseguenza andrà a costruire la tupla secondo questo schema:

< id sensore (char), valore misura (int) >

L'id del sensore può essere il carattere '***S'*** oppure ***'D'*** (rispettivamente sinistra e destra).

Nel momento in cui l'*admin* osserva un valore sopra alla soglia inserirà una particolare tupla in cui sarà contenuta l'azione che si dovrà effettuare e che sarà poi prelevata ed inoltrata agli attuatori dall'agente *output master*.
La tupla formata dall'*admin* nel momento in cui rileva un andamento non idoneo, è così fatta:

< id sensore (char), comando (char) >

Anche in questo caso, *l'id* sarà o ***'S'*** oppure ***'D'*** mentre il *comando* sarà indicato da un carattere che potrà essere ***'O'*** oppure ***'C'***, il primo indica *open*, il secondo, *close*.

L'id all'interno della tupla di comando è necessario affinché si chiuda una determinata parte del condotto. Infatti, è possibile che solamente un sensore dia un valore di gas sopra alla soglia e quindi si va a chiudere solamente una porzione del condotto e non la sua interezza.
Lo stesso ragionamento si effettua per l'apertura che sarà svolta solamente sulla porzione di condotto il cui sensore associatogli indichi un valore sotto soglia, se la restante porzione si trova ancora in allarme essa non potrà essere aperta.

Descrizione delle classi Java

a. Input Master

L'*input master*, è l'agente che ha il compito di prelevare le misurazioni dal microcontrollore, di seguito viene riportato il codice ed una breve descrizione del suo funzionamento.

```
1.  package java_proj.input_master;
2.  import java_proj.Globals;
3.  import java.io.InputStream;
4.  import lights.*;
5.  import lights.interfaces.*;
6.
7.
8.  public class Measurements_Agent extends Thread {
9.
10.     private int getIntMeasureValue(String s){
11.         int val=-1;
12.         String str_num="";
13.         s=s.trim();
14.         for(int i = 0; i< s.length(); i++){
15.             if(s.charAt(i)>='0' && s.charAt(i)<='9'){
16.                 str_num += s.charAt(i);
17.             }
18.         }
19.
20.         if(str_num != "")  val = Integer.parseInt(str_num);
21.         return val;
22.     }
23.
24.     public void run(){
25.         System.out.println("Measurements thread is running...");
26.         String measure_str="";
27.         char  char_read = ' ';
28.         char id_sensore = ' ';
29.         int measure;
30.         //int result;
31.         try {
32.             while (true)
33.             {
34.                 //while (Globals.sp.bytesAvailable() < 5)
35.                 while (Globals.sp_input.bytesAvailable() == 0)
36.                     Thread.sleep(20);
37.                 InputStream in = Globals.sp_input.getInputStream();
38.                 char_read = (char) in.read();
39.
40.                 if(char_read == Globals.ID_DX || char_read == Globals.ID_SX){
41.                     id_sensore = char_read;
42.                 }
43.
44.                 if(char_read >= '0' && char_read <= '9'){
45.                     measure_str = measure_str + char_read;
46.                 }
47.
48.                 if(char_read == Globals.END_MSG){
49.                     measure = getIntMeasureValue(measure_str.trim());
50.                     measure_str = "";
51.
52.
53.
54.                     //imposto i campi della tupla
```

```
55.                     if(measure != -1) {
56.                         IField f1 = new Field().setValue(id_sensore);
57.                         IField f2 = new Field().setValue(measure);
58.                         //creo la tupla
59.                         ITuple t1 = new Tuple();
60.                         //inserisco i campi nella tupla
61.                         t1.add(f1);
62.                         t1.add(f2);
63.                         //pubblico la tupla
64.                         System.out.println("pubblico la tupla : <" + id_sensore + " , " +
    measure + ">");
65.                         try {
66.                             Globals.ts.out(t1);
67.                         } catch (Throwable err) {
68.                             err.printStackTrace();
69.                         }
70.                     }
71.                 }
72.                 in.close();
73.             }
74.         } catch (Exception e) { e.printStackTrace(); }
75.         Globals.sp_input.closePort();
76.     }
77. }
```

Tale agente si mette in ascolto della linea seriale che verrà aperta dalla classe *main*, dalla quale proverranno i messaggi del microcontrollore contenenti l'id del sensore e la misura ad esso associata.

Una volta estrapolato l'id ed il corrispettivo valore della misura (riga 40 e 44), quando il programma legge il terminatore di messaggio indicato dalla variabile *Globals.END* (carattere '_'), andrà a costruire la tupla < id sensore (char), valore misura (int) > e la pubblicherà nello spazio di tuple chiamato *Globals.ts*.

b. Tuple Space Admin

Lo spazio di tuple, come detto prima, viene amministrato dal seguente agente software

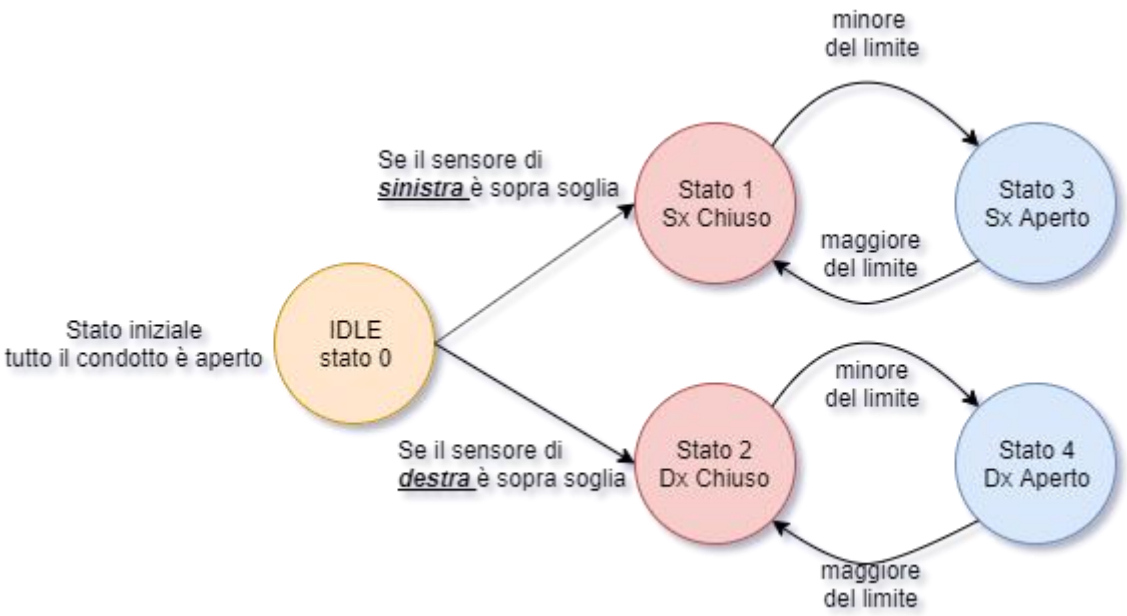
```
1.  package java_proj.tuple_space_admin;
2.  import java_proj.Globals;
3.  import java.io.InputStream;
4.  import lights.*;
5.  import lights.interfaces.*;
6.
7.  public class Tuple_Space_Admin extends Thread {
8.
9.      private void scriviTupla(char id, char command){
10.         IField f1 = new Field().setValue(id);
11.         IField f2 = new Field().setValue(command);
12.         //creo la tupla
13.         ITuple t1 = new Tuple();
14.         //inserisco i campi nella tupla
15.         t1.add(f1);
16.         t1.add(f2);
17.         //pubblico la tupla
18.         try{
19.             Globals.ts.out(t1);
20.         }catch(Throwable err){
21.             err.printStackTrace();
22.         }
23.     }
24.
```

```
25.     public void run() {
26.         System.out.println("TupleSpaceAdmin thread is running...");
27.
28.         int val_letto =-1;
29.         char id_sensore =' ';
30.
31.         IField f1 = new Field().setType(Character.class);
32.         IField f2 = new Field().setType(Integer.class);
33.         //creo la tupla
34.         ITuple p = new Tuple();
35.         //inserisco i campi nella tupla
36.         p.add(f1);
37.         p.add(f2);
38.         ITuple actual_tuple;
39.         int stato_sistema = -1;
40.         while (true) {
41.             try {
42.                 actual_tuple = Globals.ts.in(p);
43.                 val_letto = Integer.parseInt(actual_tuple.get(1).toString());
44.                 id_sensore = actual_tuple.get(0).toString().charAt(0);
45.                 System.out.println(actual_tuple + "  " + actual_tuple.get(0) + "  " + actu
    al_tuple.get(1));
46.
47.                 //implemento la logica di cambio di stato
48.                 //se supera la soglia il sensore di sinistra
49.                 if(val_letto > Globals.MAX_MEASURE_VALUE && id_sensore == Globals.ID_SX){

50.                     System.out.println("attento valore soglia sensore sx superata");
51.                     System.out.println("sensore : " + id_sensore);
52.                     stato_sistema = 1;
53.                 }
54.
55.                 //se supera la soglia il sensore di destra
56.                 if(val_letto > Globals.MAX_MEASURE_VALUE && id_sensore == Globals.ID_DX){

57.                     System.out.println("attento valore soglia sensore dx superata");
58.                     System.out.println("sensore : " + id_sensore);
59.                     stato_sistema = 2;
60.                 }
61.
62.                 //se è inferiore alla soglia il sensore di sinistra
63.                 if(val_letto < Globals.MAX_MEASURE_VALUE && id_sensore == Globals.ID_SX){

64.                     System.out.println("sensore sx okay");
65.                     stato_sistema = 3;
66.                 }
67.
68.                 //se è inferiore alla soglia il sensore di destra
69.                 if(val_letto < Globals.MAX_MEASURE_VALUE && id_sensore == Globals.ID_DX){

70.                     System.out.println("sensore dx okay");
71.                     stato_sistema = 4;
72.                 }
73.
74.                 switch (stato_sistema){
75.                     case 1:
76.                         //pubblico la tupla per far chiudere il led a sx ed il servomotore

77.                         scriviTupla(Globals.ID_SX,Globals.CLOSE);
78.                         break;
79.                     case 2:
80.                         //pubblico la tupla per far chiudere il led a dx ed il servomotore
81.                         scriviTupla(Globals.ID_DX,Globals.CLOSE);
82.                         break;
83.
84.
85.
```

```
86.                     case 3:
87.                         //pubblico la tupla per far aprire il led a sx ed il servomotore
88.                         scriviTupla(Globals.ID_SX,Globals.OPEN);
89.                         break;
90.
91.                     case 4:
92.                         //pubblico la tupla per far aprire il led a dx ed il servomotore
93.                         scriviTupla(Globals.ID_DX,Globals.OPEN);
94.                         break;
95.                 }
96.
97.             } catch (Throwable err) {
98.                 err.printStackTrace();
99.             }
100.            }
101.        }
102.    }
```

Questo è l'unico agente a non aver il bisogno di interfacciarsi con l'interfaccia seriale, bensì ha solamente lo scopo di amministrare leggere ed inserire nuove tuple.
Da questo codice e dal relativo schema a blocchi è possibile vedere che lo spazio delle tuple è amministrato da un singolo agente, e questo potrebbe essere un bottleneck nel momento in cui molti agenti andranno a scrivere ed utilizzare tale spazio di tuple, ma ai fini del progetto presentato tale architettura non presenta delle problematiche in termini di performance.

Dal codice è possibile notare l'implementazione ed il funzionamento logico della *macchina a stati*, che viene riepilogato in questo schema:



Come è possibile notare, si ha uno stato iniziale "0" in cui il sistema si trova per la prima volta in cui inizia il suo funzionamento, in cui tutto il condotto è aperto e libero da impedimenti.
Tale agente si sottoscriverà a delle tuple formate da un <*char* , *int*> in cui sarà contenuto l'id del sensore e la sua misura associata.
Nel momento in cui legge un valore maggiore della soglia cambia lo stato e va a chiudere la rispettiva porzione del condotto.
In questo stato l'agente andrà ad inserire una tupla che indicherà il comando che dovrà essere eseguito, formata in questo modo

<*char, char*>

c. Output Agent

Questo è l'agente che ha il compito di andare a fornire i comandi al microcontrollore che comanda gli attuatori.

```
1.  package java_proj.output_master;
2.
3.  import java_proj.Globals;
4.  import lights.Field;
5.  import lights.Tuple;
6.  import lights.interfaces.IField;
7.  import lights.interfaces.ITuple;
8.
9.  public class Output_Agent extends Thread {
10.
11.     private static final char chiudi_sx = 'A'; // chiude il servomotore ed il led sx
12.     private static final char chiudi_dx = 'B'; // chiude il servomotore ed il led dx
13.
14.     private static final char chiudi_all = 'C'; // chiude il servomotore, il led dx & sx
15.
16.     private static final char apri_sx = 'X'; // apre il servomotore ed il led sx
17.     private static final char apri_dx = 'Y'; // apre il servomotore ed il led dx
18.
19.     private static final  int OPEN = 0; // led verde sx/dx acceso & servomotore aperto
20.     private static final  int CLOSE = 1; //led rosso sx/dx acceso & servomotore chiuso
21.
22.
23.     private void serialWrite(char cmd) {
24.         try{
25.             Globals.sp_output.getOutputStream().write(cmd);
26.             //System.out.println("Sent Command: " + cmd);
27.             Globals.sp_output.getOutputStream().flush();
28.         }catch (Throwable err){
29.             System.out.println("Impossibile scrivere sulla linea seriale");
30.             err.printStackTrace();
31.         }
32.     }
33.
34.
35.     public void run() {
36.         System.out.println("OutputAgent thread is running...");
37.         int stato_sx, stato_dx;
38.         stato_dx = stato_sx = OPEN;
39.         int flag = 0;
40.
41.         IField f1 = new Field().setType(Character.class);
42.         IField f2 = new Field().setType(Character.class);
43.         char id,cmd;
44.         //creo la tupla
45.         ITuple p = new Tuple();
46.         //inserisco i campi nella tupla
47.         p.add(f1);
48.         p.add(f2);
49.         ITuple actual_tuple;
50.         while(true){
51.
52.             try {
53.                 actual_tuple = Globals.ts.in(p);
54.                 id = actual_tuple.get(0).toString().charAt(0);
55.                 cmd = actual_tuple.get(1).toString().charAt(0);
56.                 System.out.println(actual_tuple + "  " + id + "  " + cmd);
57.
58.
```

```
59.                 if (flag == 0){
60.                     //se è la prima volta che esegue
61.                     stato_dx = stato_sx = OPEN;
62.                     flag = 1;
63.                 }
64.
65.                 if(id == Globals.ID_DX && cmd == Globals.OPEN && stato_dx == CLOSE){
66.                     /* se ho un messaggio per il sensore di destra e la soglia è inferiore

67.                      apro il servomotore ed il led di destra se esso non è già chiuso*/
68.                     stato_dx = OPEN;
69.                     serialWrite(apri_dx);
70.                 }
71.                 if(id == Globals.ID_DX && cmd == Globals.CLOSE && stato_dx ==OPEN){
72.                     /* se ho un messaggio per il sensore di destra e la soglia è maggiore
73.                      chiudo il servomotore ed il led di destra se esso non è già chiuso*/

74.                     stato_dx = CLOSE;
75.                     serialWrite(chiudi_dx);
76.                 }
77.
78.
79.
80.                 if(id == Globals.ID_SX && cmd == Globals.OPEN && stato_sx == CLOSE){
81.                     /* se ho un messaggio per il sensore di sinistra e la soglia è inferio
    re
82.                      apro il servomotore ed il led di sinistra se esso non è già chiuso*/

83.                     stato_sx = OPEN;
84.                     serialWrite(apri_sx);
85.                 }
86.
87.                 if(id == Globals.ID_SX && cmd == Globals.CLOSE && stato_sx == OPEN){
88.                     /* se ho un messaggio per il sensore di sinistra e la soglia è maggior
    e
89.                      chiudo il servomotore ed il led di sinistra se esso non è già chiuso*
    /
90.                     stato_sx = CLOSE;
91.                     serialWrite(chiudi_sx);
92.                 }
93.
94.             }catch (Throwable err) {
95.                 err.printStackTrace();
96.             }
97.         }
98.     }
99. }
```

Tale agente si mette in ascolto sullo spazio delle tuple e sottoscrive il suo interesse a tutte quelle così formate : *<char, char>*
Infatti, tali tuple, conterranno i comandi da dover inoltrare, tramite linea seriale, al microcontrollore.
L'agente tiene in memoria lo stato del sistema e va ad inoltrare il messaggio solamente se legge un comando in contraddizione con lo stato attuale.
In altre parole, l'agente per prima cosa, al momento della sua prima esecuzione, sa che tutte le paratie sono aperte, quindi andrà ad inoltrare il messaggio al microcontrollore solamente se dallo spazio di tuple leggerà il messaggio di chiusura.
Allo stesso modo se l'agente è al corrente che una porzione del condotto è chiusa, andrà ad inoltrare al microcontrollore il comando di apertura quando leggerà la tupla corrispondente a tale comando, altrimenti non effettua nulla e si resterà in attesa.

Script Arduino

Riporto ora i due script che sono eseguiti dalle due schede Arduino, il primo che viene descritto è quello a cui sono collegati i sensori di gas che andrà esclusivamente ad interagire con l'*input agent*.

a. Sensor Sketch

```
#define DIGITAL_IN_SX 3
#define DIGITAL_IN_DX 2
#define ANALOG_IN_DX A0
#define ANALOG_IN_SX A3

float gas_value;
char buffer[8];

void scrivi_seriale(char id_sens, int value){
  buffer[0]='\0';
  sprintf(buffer,"%c%d%c",id_sens,value,'_');
  Serial.write(buffer,sizeof(char)*8);

}

void allarm_dx(){
  float gas_value=analogRead(ANALOG_IN_DX);
  delay(500);
  scrivi_seriale('D',gas_value);
}

void allarm_sx(){
  float gas_value=analogRead(ANALOG_IN_SX);
  delay(500);
  scrivi_seriale('S',gas_value);

}

void setup()
{
  pinMode(ANALOG_IN_DX,INPUT);
  pinMode(ANALOG_IN_SX,INPUT);
  attachInterrupt(digitalPinToInterrupt(DIGITAL_IN_DX), allarm_dx, FALLING);
  attachInterrupt(digitalPinToInterrupt(DIGITAL_IN_SX), allarm_sx, FALLING);

  Serial.begin(9600);
}

void loop(){
  if(digitalRead(DIGITAL_IN_DX) == HIGH) scrivi_seriale('D',0);

  delay(2000);

  if(digitalRead(DIGITAL_IN_SX) == HIGH) scrivi_seriale('S',0);

  delay(2000);
}
```

Dal codice qui riportato, è possibile vedere che ogni due secondi il microcontrollore, se tutto funziona correttamente, invia un valore nullo.

Grazie all'uscita digitale del sensore si può applicare una logica ad eventi, infatti, ai pin del microcontrollore 2 e 3, saranno collegati degli interrupt. Quando il segnale passa ad un valore logico nullo, significa che il sensore ha rilevato la presenza di gas e questo va a scatenare un evento ed il microcontrollore andrà a scrivere sulla linea seriale il valore del gas in ppm.
Altrimenti se il livello logico dell'uscita digitale è a livello alto il microcontrollore invierà 0.

b. Actuator Sketch

```
#include <Servo.h>

#define LED_ROSSO_SX 3
#define LED_VERDE_SX 2

#define LED_ROSSO_DX 4
#define LED_VERDE_DX 5

#define PIN_SERVO 9
#define PIN_SWITCH_SERVO 12

#define CHIUSO 0
#define APERTO 1

char chiudi_sx = 'A'; // chiude il servomotore ed il led sx
char chiudi_dx = 'B'; // chiude il servomotore ed il led dx

char apri_sx = 'X'; // apre il servomotore ed il led sx
char apri_dx = 'Y'; // apre il servomotore ed il led dx


Servo myservo;
int pos, flag_servo, switch_servo;
volatile int stato = 0,statoDx, statoSx;
volatile int futureState=0;
char incomingByte ="";

void apri_valvola() {

  if (flag_servo == 1 ) {
    flag_servo = 0;
    for (pos = pos; pos >= 0; pos--) {
      myservo.write(pos);
      delay(15);
    }
  }
  return;

}

void chiudi_valvola() {

  if (flag_servo == 0 ) {
    flag_servo = 1;
    for (pos = pos; pos < 90; pos++) {
      myservo.write(pos);
      delay(15);
    }
  }
  return;
}

```

```
53. //--------------- VERDE ----------------------------------
54. void accendiVerdeDx() {
55.   digitalWrite(LED_VERDE_DX, HIGH);
56.   return;
57. }
58.
59. void accendiVerdeSx() {
60.   digitalWrite(LED_VERDE_SX, HIGH);
61.   return;
62. }
63.
64. void spegniVerdeDx() {
65.   digitalWrite(LED_VERDE_DX, LOW);
66.   return;
67. }
68.
69. void spegniVerdeSx() {
70.   digitalWrite(LED_VERDE_SX, LOW);
71.   return;
72. }
73.
74. //--------------- ROSSO ----------------------------------
75.
76. void accendiRossoDx() {
77.   digitalWrite(LED_ROSSO_DX, HIGH);
78.   return;
79. }
80.
81. void accendiRossoSx() {
82.   digitalWrite(LED_ROSSO_SX, HIGH);
83.   return;
84. }
85.
86. void spegniRossoDx() {
87.   digitalWrite(LED_ROSSO_DX, LOW);
88.   return;
89. }
90.
91. void spegniRossoSx() {
92.   digitalWrite(LED_ROSSO_SX, LOW);
93.   return;
94. }
95. //------------------------------------------------------------
96. void setup() {
97.          pinMode(LED_ROSSO_SX, OUTPUT);
98.          pinMode(LED_ROSSO_DX, OUTPUT);
99.          pinMode(LED_VERDE_SX, OUTPUT);
100.         pinMode(LED_VERDE_DX, OUTPUT);
101.         Serial.begin(9600);
102.         myservo.attach(PIN_SERVO);
103.         myservo.write(0);
104.         pos = 0;
105.         flag_servo = 0;
106.         statoDx = statoSx = 0;
107.         incomingByte="";
108.       }
109.
110.      void loop() {
111.        if (stato == 0) {
112.          accendiVerdeDx();   // accende il LED VERDE A SX
113.          accendiVerdeSx();    // accende il LED VERDE A DX
114.          spegniRossoDx();   // accende il LED VERDE A SX
115.          spegniRossoSx();    // accende il LED VERDE A DX
116.          apri_valvola();
117.          statoDx = statoSx = APERTO;
118.          stato = 1;
119.        }
```

```
120.         incomingByte = Serial.read();
121.         if (stato != 0 && incomingByte !=""){
122.           // put your main code here, to run repeatedly:
123.
124.           if(incomingByte == chiudi_sx && statoSx == APERTO) {
125.             //chiudo il servomotore e spengo il led verde a sinistra ed accendo il rosso

126.             chiudi_valvola();
127.             spegniVerdeSx();
128.             accendiRossoSx();
129.             statoSx = CHIUSO;
130.           }
131.
132.           if(incomingByte == chiudi_dx && statoDx == APERTO) {
133.             //chiudo il servomotore e spengo il led verde a sinistra ed accendo il rosso

134.             chiudi_valvola();
135.             spegniVerdeDx();
136.             accendiRossoDx();
137.             statoDx = CHIUSO;
138.           }
139.
140.           if(incomingByte == apri_sx && statoSx == CHIUSO){
141.             if(statoDx== APERTO) apri_valvola();
142.             spegniRossoSx();
143.             accendiVerdeSx();
144.             statoSx = APERTO;
145.           }
146.
147.           if(incomingByte == apri_dx && statoDx == CHIUSO){
148.             if(statoSx== APERTO) apri_valvola();
149.             spegniRossoDx();
150.             accendiVerdeDx();
151.             statoDx = APERTO;
152.           }
153.
154.           if(statoDx == APERTO && statoSx == APERTO){
155.             apri_valvola();
156.           }
157.
158.
159.         }
160.
161.       }
```

Dal programma appena riportato è possibile il funzionamento e la logica della gestione degli attuatori. Nelle prima righe di codice sono riporti i caratteri chiave che corrispondono ad un determinato comando, che sono:

- A : chiude il servomotore ed il led sx
- B : chiude il servomotore ed il led dx
- X : apre il servomotore ed il led sx
- Y : apre il servomotore ed il led dx

Tali caratteri sono memorizzati all'interno di variabili in modo da rendere una futura modifica più semplice.
Nel momento in cui il microcontrollore legge dalla seriale uno di questi quattro comandi va a cambiare lo stato delle paratie, è bene notare che tale stato viene modificato solamente se il comando in entrata è diverso da quello attualmente in essere.
Inoltre la paratia centrale verrà aperta solamente se quella di sinistra e destra sono libere, infatti questo è un modellino, ma nella parte a destra e sinistra potrebbero esserci altre diramazioni e quindi è bene

aprire il condotto solamente se le restanti parti possono essere aperte. In questo modo non si preclude il corretto funzionamento della porzione della condotta.

Conclusioni & Precisazioni progettuali

Grazie all'uso dello spazio di tuple è possibile disaccoppiare gli agenti dal loro compito, inoltre non è necessario che gli agenti si conoscano a vicenda. Infatti, gli agenti devono solamente conoscere ed avere il riferimento all'oggetto dello spazio di tuple e sapere la logica con il quale vengono scritte le tuple.
In questo modo gli agenti che gestiscono i microcontrollori possono essere dislocati e non per forza risiedere sullo stesso punto di esecuzione, che nella presentazione è il laptop.

Il prototipo presentato usa una nomenclatura, per i sensori, che non è del tutto idonea dal punto di vista dei sistemi distribuiti. Infatti, utilizzando la nomenclatura *destra* e *sinistra* si preclude un funzionamento più flessibile, infatti se in una porzione di condotto si va ad aggiunge un altro sensore, viene meno il significato delle nomenclature con l'obbligo quindi di cambiare sia lo script eseguito da arduino e sia i nomi delle variabili del programma Java.
Per facilitare però tale modifica si è scelto di creare una classe chiamata *Global*

```
1.  public class Globals {
2.      public static SerialPort sp_input; // definizione della porta seriale di ingresso
3.      public static SerialPort sp_output; // definizione della porta seriale di ingresso
4.
5.      public static ITupleSpace ts = new TupleSpace("Tuple_space"); //spazio delle tuple
6.      public static final int MAX_MEASURE_VALUE = 20; // massimo valore della misura di gas
7.
8.      public static final char ID_SX = 'S'; // id del sensore di sinistra
9.      public static final char ID_DX = 'D'; // id del sensore di destra
10.     public static final char END_MSG = '_'; // terminatore del messaggio
11.
12.     // variabili per il comando della chiusura delle paratie
13.     public static final char OPEN = 'O';// apre il condotto e quindi mette il led a verde e
    d apre il servomotore
14.     public static final char CLOSE = 'C'; // chiude il condotto e quindi mette il led a ros
    so e chiude il servomotore
15. }
```

Nella quale sono definiti tutti i comandi e gli ID usati dal programma in modo da avere una facile modifica.

Una futura implementazione potrebbe essere quella di associare un microcontrollore economico per ogni sensore, il quale invierà la misurazione con un ID univoco, il sistema centrale, essendo a conoscenza di dove è posizionato geograficamente tale ID andrà a scrivere la tupla per la parte di tubazione associata a tale sensore.
In questo modo si renderebbe facile la sostituzione del sensore poiché è solamente il software della scheda elettronica a farsi carico della scelta dell'ID.
Inoltre, così facendo, si semplifica l'aggiunta di altri sensori in un tratto di linea, basta inserire un record nella tabella che associa gli ID alla loro posizione.