Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria “Enzo Ferrari”

Corso di Laurea Magistrale in Ingegneria Informatica



Modellazione e prototipazione di un sensore 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

[Sommario 2](#_Toc29306537)

[Introduzione 3](#_Toc29306538)

[a. Finalità del progetto 3](#_Toc29306539)

# Introduzione

# 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.

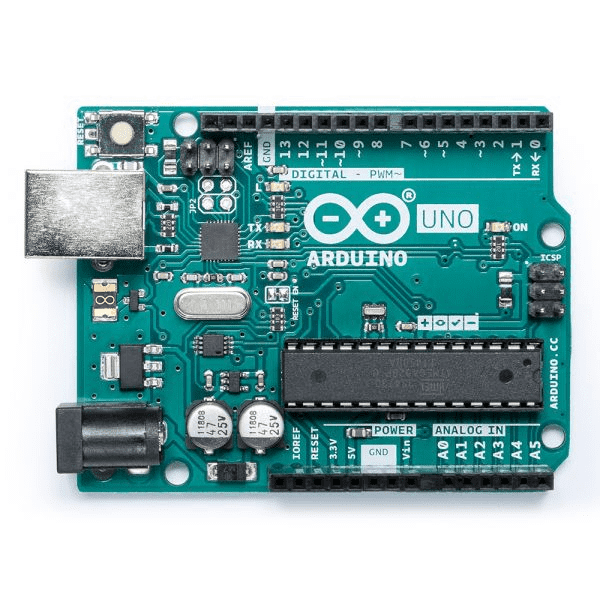
Il sistema è controllato da due microcontrollori, uno ha il compito di prelevare i dati, le misure, dai sensori di gas posti al di sopra della condotta, mentre l’altro, ha lo scopo di andare ad azionare le paratie in base ai comandi provenienti dal programma Java che si interfaccerà con questi due microcontrollori 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.

# 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 Pins6

## \* 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).

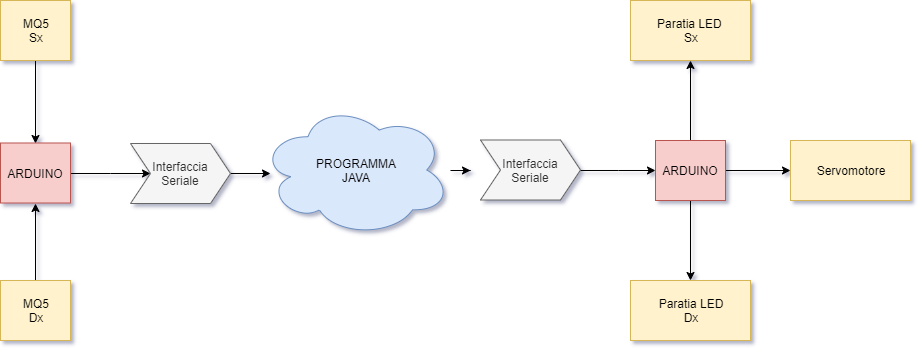
L’uscita digitale assume valore logico

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.

# Schema a blocchi complessivo



Dallo schema a blocchi sopra, è possibile capire l’architettura del progetto e come i componenti sono interfacciati tra loro.

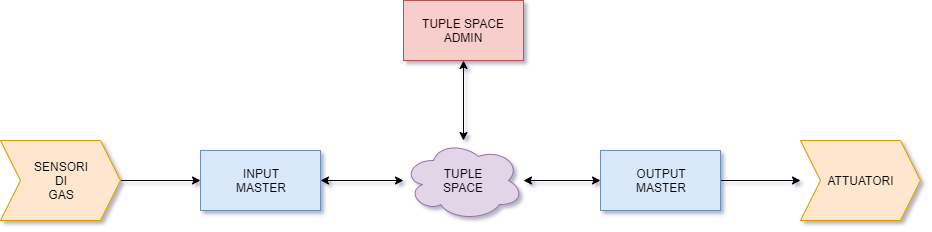
Per prima cosa si può vedere che il programma JAVA “dialoga” con i due microcontrollori con il protocollo seriale, dal verso delle frecce è possibile inoltre capire come avviene la comunicazione.

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.

# 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.

# Input Master

Riporto ora il codice dell’agente che ha il compito di prelevare le misurazioni ed a seguito verrà effettuata una breve descrizione dei passaggi più importanti.

1. **package** java\_proj.input\_master;
2. **import** java\_proj.Globals;
3. **import** java.io.InputStream;
4. **import** lights.\*;
5. **import** lights.interfaces.\*;

8. **public** **class** Measurements\_Agent **extends** Thread {
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. }
20. **if**(str\_num != "")  val = Integer.parseInt(str\_num);
21. **return** val;
22. }
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();
40. **if**(char\_read == Globals.ID\_DX || char\_read == Globals.ID\_SX){
41. id\_sensore = char\_read;
42. }
44. **if**(char\_read >= '0' && char\_read <= '9'){
45. measure\_str = measure\_str + char\_read;
46. }
48. **if**(char\_read == Globals.END\_MSG){
49. measure = getIntMeasureValue(measure\_str.trim());
50. measure\_str = "";
52. //imposto i campi della tupla
53. **if**(measure != -1) {
54. IField f1 = **new** Field().setValue(id\_sensore);
55. IField f2 = **new** Field().setValue(measure);
56. //creo la tupla
57. ITuple t1 = **new** Tuple();
58. //inserisco i campi nella tupla
59. t1.add(f1);
60. t1.add(f2);
61. //pubblico la tupla
62. System.out.println("pubblico la tupla : <" + id\_sensore + " , " + measure + ">");
63. **try** {
64. Globals.ts.out(t1);
65. } **catch** (Throwable err) {
66. err.printStackTrace();
67. }
68. }
69. }
70. in.close();
71. }
72. } **catch** (Exception e) { e.printStackTrace(); }
73. Globals.sp\_input.closePort();
74. }
75. }

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*.

# 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.\*;
7. **public** **class** Tuple\_Space\_Admin **extends** Thread {
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. //System.out.println("pubblico la tupla");
19. **try**{
20. Globals.ts.out(t1);
21. }**catch**(Throwable err){
22. err.printStackTrace();
23. }
24. }
26. **public** **void** run() {
27. System.out.println("TupleSpaceAdmin thread is running...");
29. **int** val\_letto =-1;
30. **char** id\_sensore =' ';
32. IField f1 = **new** Field().setType(Character.**class**);
33. IField f2 = **new** Field().setType(Integer.**class**);
34. //creo la tupla
35. ITuple p = **new** Tuple();
36. //inserisco i campi nella tupla
37. p.add(f1);
38. p.add(f2);
39. ITuple actual\_tuple;
40. **int** stato\_sistema = -1;
41. **while** (**true**) {
42. **try** {
43. actual\_tuple = Globals.ts.in(p);
44. val\_letto = Integer.parseInt(actual\_tuple.get(1).toString());
45. id\_sensore = actual\_tuple.get(0).toString().charAt(0);
46. System.out.println(actual\_tuple + "  " + actual\_tuple.get(0) + "  " + actual\_tuple.get(1));
48. //implemento la logica di cambio di stato
49. //se supera la soglia il sensore di sinistra
50. **if**(val\_letto > Globals.MAX\_MEASURE\_VALUE && id\_sensore == Globals.ID\_SX){
51. System.out.println("attento valore soglia sensore sx superata");
52. System.out.println("sensore : " + id\_sensore);
53. stato\_sistema = 1;
54. }
56. //se supera la soglia il sensore di destra
57. **if**(val\_letto > Globals.MAX\_MEASURE\_VALUE && id\_sensore == Globals.ID\_DX){
58. System.out.println("attento valore soglia sensore dx superata");
59. System.out.println("sensore : " + id\_sensore);
60. stato\_sistema = 2;
61. }
63. //se è inferiore alla soglia il sensore di sinistra
64. **if**(val\_letto < Globals.MAX\_MEASURE\_VALUE && id\_sensore == Globals.ID\_SX){
65. System.out.println("sensore sx okay");
66. stato\_sistema = 3;
67. }
69. //se è inferiore alla soglia il sensore di destra
70. **if**(val\_letto < Globals.MAX\_MEASURE\_VALUE && id\_sensore == Globals.ID\_DX){
71. System.out.println("sensore dx okay");
72. stato\_sistema = 4;
73. }
75. **switch** (stato\_sistema){
76. **case** 1:
77. //pubblico la tupla per far chiudere il led a sx ed il servomotore
78. scriviTupla(Globals.ID\_SX,Globals.CLOSE);
79. **break**;
80. **case** 2:
81. //pubblico la tupla per far chiudere il led a dx ed il servomotore
82. scriviTupla(Globals.ID\_DX,Globals.CLOSE);
83. **break**;
85. **case** 3:
86. //pubblico la tupla per far aprire il led a sx ed il servomotore
87. scriviTupla(Globals.ID\_SX,Globals.OPEN);
88. **break**;
89. **case** 4:
90. //pubblico la tupla per far aprire il led a dx ed il servomotore
91. scriviTupla(Globals.ID\_DX,Globals.OPEN);
92. **break**;
93. }
95. } **catch** (Throwable err) {
96. err.printStackTrace();
97. }
98. }
99. }
100. }

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.