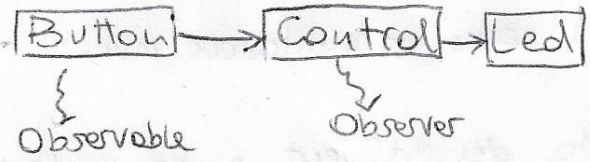
INTRO -1-

OBIETTIVO: Costruire applicazioni IOT come semplici sist composti da SENSORI e ATTUATORI ①

↓ Button ↓ Led

ARCHITETTURA:

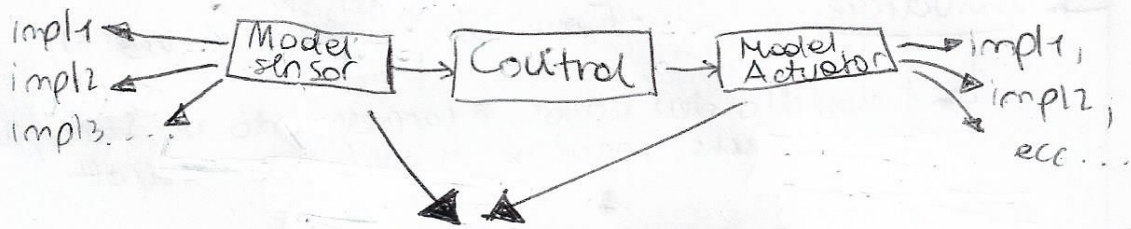


Il codice dell'appl. non può essere responsabilità né del Button né del Led, quindi c'è un CONTROL

- es
  - ROa: Button premuto → blink → premuto → no blink
  - Rob: Temperatura sopra una soglia → Led on, sennò off

Dobbiamo considerare un set di attuatori e sensori in un SIST DISTR

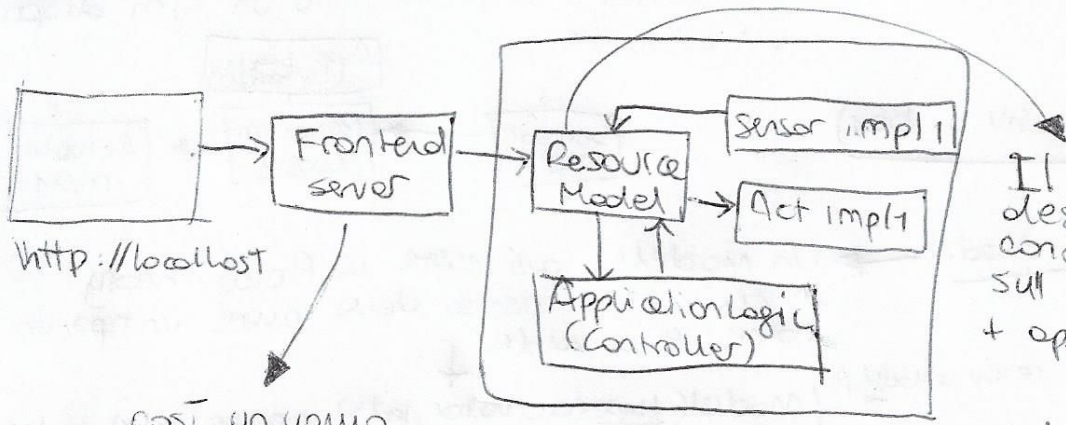
DISACCOPPIAMENTO DAI DETTAGLI TECN



ogni sensore/attuat dà la sua implementaz. x modificare il modello → disaccopp. tra CONTROLLER (Business logic)

DETTAGLI TECNOLOGICI

FRONTEND



così un uomo può inviare comandi x modificare lo st. degli attuatori come se fosse una macchina

Il software designer dovrebbe concentrare l'atten. sul Resource Model + appropr.

e delegare al prossimo step architetturale
- i legami tra il modello e i dispositivi concreti

SYSTEM MODELS -2-

→ Uso dei Mock → GUI-based components

→ Definisce un sistema bls1 x il req ROb con gli eventi sensorEvent e ctrlEvent

- il qasensor è modellato come un'entità che invia sensorEvent
- il qaactuator invece aspetta dei ctrlEvent e usa un Led Mock x accenderlo quando l'evento è ctrlEvent(on) con javaRun e viceversa
- il qacontrol aspetta sensorEvent e se la temp. è troppo alta invia ctrlEvent(on) senó ctrlEvent(off)

  ↓ la business logica è nel qacontrol sottoforma di regole Prolog

→ x il req boa invece:

→ 2 componenti:

- attuatore
  - Dispatch turn : switch
  - Event local_click : clicked(N)
  - il modello dell attuat. è rappresentato da un fatto, ma ci sono anche regole x modificare on/off

    ↓ quando arriva il dispatch turn esegue una regola x switchare da on/off
- controller → crea un Button Mock che emette local_click e quando arrivano invia un turn dispatch all'attuatore

MVC (relenv e bo1)

→ Resource Model → Un modello può essere la Prolog Theory in cui ogni risorsa deve avere un tipo, un nome e un valore

↓

resourceModel.pl | model(type(actuator, leds), name(led1), value(off))
model(type(sensor, temperature), name(t1), value(25)).

↓ ultimamente si usa JSON, ma con Prolog si possono usare regole x ottenere o modificare il modello

il sist definisce eventi e eventhandler

CONTROLLER

i sensorEvent emessi dal disp. di temperatura t1 sono mappati come inputCtrlEvent

quando arriva un inputCtrlEvent, cambia il modello con la regola changeModelItem di ResourceModel.pl

In particolare specifica nelle Rules changedModelAction (che era in fondo a changeModelItem in resourceModel.pl), così che venga eseguito dopo la modifica del modello.

In sintesi:

- quando invoco un changeModelIt sulla temperatura, viene anche fatta una changedModelAction sulla temperatura, che dopo aver controllato se è alta o bassa, invoca un changeModelItem del led (x accenderlo o spegnerlo)
- un changeModelItem del led porta anche a un changeModelAction del led che a sua volta consiste in un emitevent

questo è presente nel resourceModel.pl e permette di inviare un evento all'attore corrente con un certo payload (passato per param.)

ATTUATORE

in questo caso si invia ctrlEvent

può essere un attore che quando arriva ctrlEvent discrimina in base a (leds, led1, on/off) e fa javaRun a seconda dei casi (led on o led off)