Intelligent System Project

Simone Vuotto

18 gennaio 2016

1 Introduzione

Il progetto realizzato ha l'obiettivo di utilizzare il framework zetta (`http://zettajs.org`) per la creazione di un semplice sistema ad agenti. Nel sistema realizzato sono presenti due agenti: uno in esecuzione su una Raspberry Pi, responsabile del controllo della temperatura circostante, e l'altro in esecuzione su un laptop, responsabile di tenere un log degli eventi presenti nel sistema. Infine, è stata realizzata una web app in AngularJS (`https://angularjs.org/`) che permette all'utente di interagire con il sistema.

1.1 Zetta Overview

Zetta è una piattaforma open source sviluppata in NodeJS per creare server per l' Internet of Thinks (IoT) e può essere eseguito su diversi computer geo-localizzati e nel cloud. Zetta combina REST API, WebSockets e alcuni aspetti di Reactive Programming. I suoi componenti principali sono:

- **Zetta Server:** è il più alto livello di astrazione in zetta. Esso contiene Drivers, Scouts e Apps;
- **Devices:** sono responsabili di modellare dispositivi con macchine a stati e interagire con essi. Questi modelli sono usati da zetta per generare REST API;
- **Scouts:** sono utilizzati come meccanismo di scoperta dei dispositivi che potrebbero essere connessi in rete o necessitano di eseguire particolari protocolli;
- **Apps:** permettono di creare interazioni tra diversi dispositivi per definire dinamiche più complesse e permettono la creazione di REST API slegate dai singoli dispositivi.

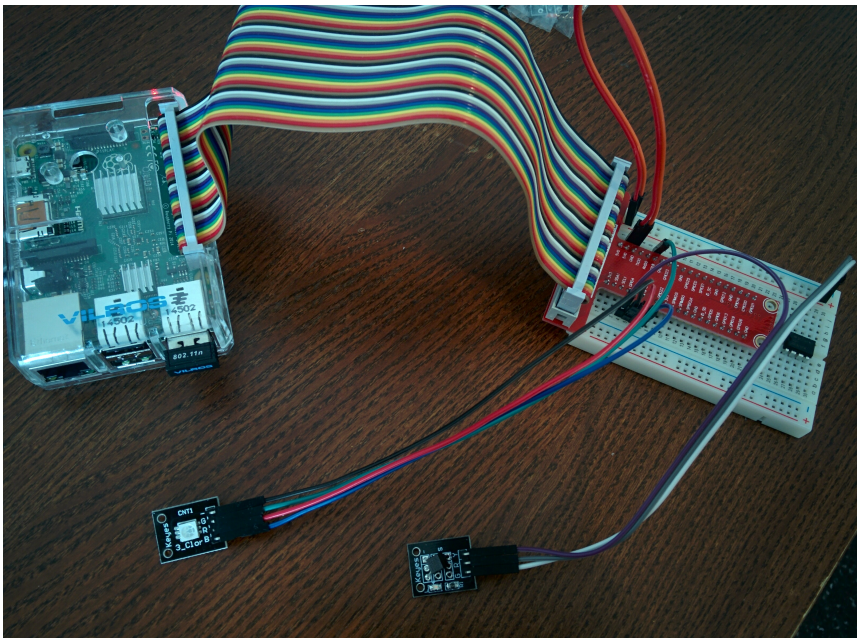

Figura 1: Foto della Raspberry Pi e dei sensori utilizzati per il progetto

2 Raspberry Pi Server

Il primo server zetta *raspberry-server* è stato installato su una Raspberry Pi 2 Model B v1.1 collegata a due sensori come mostrato in Figura 1. I due sensori sono rispettivamente:

- **LED RGB:** semplice led rgb con un pin attaccato al ground e tre pin per i colori rosso, verde e blu attaccati rispettivamente ai pin 11, 12 e 13 (physical numbering) usati utilizzando la modulazione PWM.
- **Sensore di temperatura** `DS18B20`**:** Sensore digitale di temperatura con un pin attaccato all'alimentazione (+5V), uno al Ground e uno al pin 7 (physical numbering).

2.1 Devices

2.1.1 LED RGB

La definizione delle classe Device responsabile della gestione del led RGB è contenuta nel file `/pi-app/pi-led-device.js`. La macchina a stati descritta al suo interno è rappresentata dal frammento di codice seguente:

```
1 // State Machine
2 config
```

```
3   .when('off', { allow: ['turn-on', 'set-color']})
4   .when('on', { allow: ['turn-off', 'set-color']})
5   .map('turn-off', this.turnOff)
6   .map('turn-on', this.turnOn)
7   .map('set-color', this.setColor, [{type: 'text', name:'color'}])
8   .monitor('color');
```

Listing 1: Definizione della macchina a stati del led RGB

Lo stato del dispositivo è rappresentato dalla variable `state` che può assumere i valori `on` e `off` e la variabile `color` che può assumere qualsiasi valore espresso da sei cifre esadecimali (es. #FFFFFF).

Le azioni consentite sul dispositivo sono `turn-on`, `turn-off` e `set-color`.

Per la gestione dei pin GPIO in NodeJS sono presenti molte librerie open source, ma per il progetto è stata scelta *wirig-pi* perchè è l'unica a supportare la simulazione software di pin PWM (una caratteristica necessaria per la comunicazione con il led perché ha bisogno di tre pin con modulazione PWM ma Raspberry pi nativamente fornisce solo un pin funzionante in questa modalità).

2.1.2 Sensore di temperatura

La definizione delle classe Device responsabile della gestione del led RGB è contenuta nel file `/pi-app/pi-temperature-sensor-device.js`. La macchina a stati descritta al suo interno è rappresentata dal frammento di codice seguente:

```
1  // State Machine
2  config
3    .when('on', { allow: ['turn-off']})
4    .when('off', { allow: ['turn-on']})
5    .map('turn-on', this.turnOn)
6    .map('turn-off', this.turnOff)
7    .monitor('temperature')
8    .stream('temperature-stream', this.streamTemperature);
```

Listing 2: Definizione della macchina a stati del sensore di temperatura

In questo caso lo stato è rappresentato dalle variabili `state` (on/off) e `temperature` (float) e le azioni possibili sono solo `turn-on` e `turn-off`.

Da notare che il metodo `monitor` dice a zetta di creare una nuova REST API per permettere di monitorare lo stato della variabile `temperature`, mentre il metodo `stream` crea in automatico uno stream di dati generati dalla funzione `this.streamTemperature` e lo espone pubblicamente tramite WebSocket.

Infine, per il corretto funzionamento di questo dispositivo è necessario eseguire le seguenti istruzioni prima dell'avvio del server zetta:

```
1  sudo modprobe w1-gpio
2  sudo modprobe w1-therm
```

2.2 App

L'app contenuta in `/pi-app/pi-app.js` definisce altre due api per leggere e scrivere il valore della variabile `threshold`. L'app utilizza questa variabile per

controllare il valore letto dal sensore di temperatura e modificare lo stato del led di conseguenza: se la temperatura è sopra soglia il led viene acceso, viceversa viene spento. Inoltre, se *raspberry-server* e *logger-server* sono in contatto tra loro (se la connessione tra i due viene persa, zetta pensa automaticamente a ristabilirla appena possibile), l'applicazione utilizza anche i dati a sua disposizione per scrivere nel log dei messaggi di diagnostica a intervalli regolari (ogni 10 secondi se la temperatura è sotto soglia, ogni 3 se la soglia viene superata) e un messaggio ogni volta che la threshold viene modificata.

Fare una query dei dispositivi è particolarmente semplice con zetta:

```
var LEDQuery = server
                    .where({type: 'led'});
var TemperatureSensorQuery = server
                    .where({type: 'temperature-sensor'});
var LoggerQuery = server
                    .from('logger-server')
                    .where({type: 'logger'});
server.observe([LoggerQuery, TemperatureSensorQuery],
               function(logger, tempSensor) {
 ...
});

server.observe([LEDQuery, TemperatureSensorQuery],
               function (led, tempSensor) {
 ...
});
```

Il metodo `observe` dell'oggetto server richiama la funzione passata come secondo argomento appena i Device richiesti nell'array passato come primo argomento sono disponibili (devono essere tutti presenti nello stesso momento, altrimenti la funzione non viene chiamata).

3 Logger Server

Il server zetta *logger-server* è stato installato su un laptop e la sua funzione è quella di salvare in maniera persistente su disco tutti i messaggi che gli vengono mandati, mostrando a chi lo richiede solo gli ultimi n messaggi.

3.1 Devices

3.1.1 Logger

La definizione delle classe Device responsabile della gestione del logger è contenuta nel file `/logger-app/logger-device.js`. La macchina a stati descritta al suo interno è rappresentata dal frammento di codice seguente:

```
// State Machine
config
  .when('enabled', {allow: ['write', 'set-length', 'disable']})
  .when('disabled', {allow: ['enable']})
  .map('write', this.write, [{type: 'text', name: 'textToWrite'}])
```

```
6   .map('set-length', this.setLength, [{type: 'int', name: 'length'
        }])
7   .map('enable', this.enable)
8   .map('disable', this.disable)
9   .monitor('logs')
10  .monitor('length');
```

Listing 3: Definizione della macchina a stati del logger

Lo stato del logger è definito da tre variabili:

- `state`: indica se il logger è abilitato (*enabled*/*disabled*)
- `logs`: è un array contenente gli ultimi n messaggi ricevuti
- `length`: intero positivo che indica qual'è la lunghezza massima che l'array `logs` può avere (i messaggi più vecchi vengono scartati).

Le azioni applicabili sul logger sono: `enable`, `disable`, `write`, `set-length`.

4 Client

Il client è la web app sviluppata in AngularJS mostrata in Figura 2. E' una dashboard che l'utente può utilizzare con il proprio browser per interagire con il sistema. Esso interagisce con i due server zetta sopra menzionati effettuando chiamate asincrone alle REST API generate automaticamente da zetta.

5 Zetta Browser

Zetta mette anche a disposizione un servizio di gestione delle proprie api tramite una web app disponibile all'indirizzo `http://browser.zettajs.io/`. Il servizio chiede a quale indirizzo ip collegarsi (essendo un'applicazione javascript può collegarsi anche ad un indirizzo locale), elabora il documento json generato in automatico dal server zetta contentente la descrizione di tutti i device disponibili ed infine mostra una dashboard generica come quella mostrata in Figura 3.

Infine, Zetta fornisce anche un server proxy installabile sulla piattaforma cloud Heroku (`http://www.heroku.com`) che permette di rendere accessibili sul web le proprie API. Questo server può collegarsi ad una o più istanze di server zetta (utilizzando il metodo `link` dell'oggetto `zetta` prima della creazione del server) e consente quindi di accedere ai propri dispositivi senza conoscere il reale indirizzo IP della macchina e senza particolari configurazioni del firewall.

6 Conclusioni

Con zetta è stato possibile costruire in poco tempo un sistema ad agenti distribuito in grado di cooperare ed interagire con il mondo esterno. Questo progetto mette in mostra lo stretto legame tra sistema ad agenti (distribuito) e REST

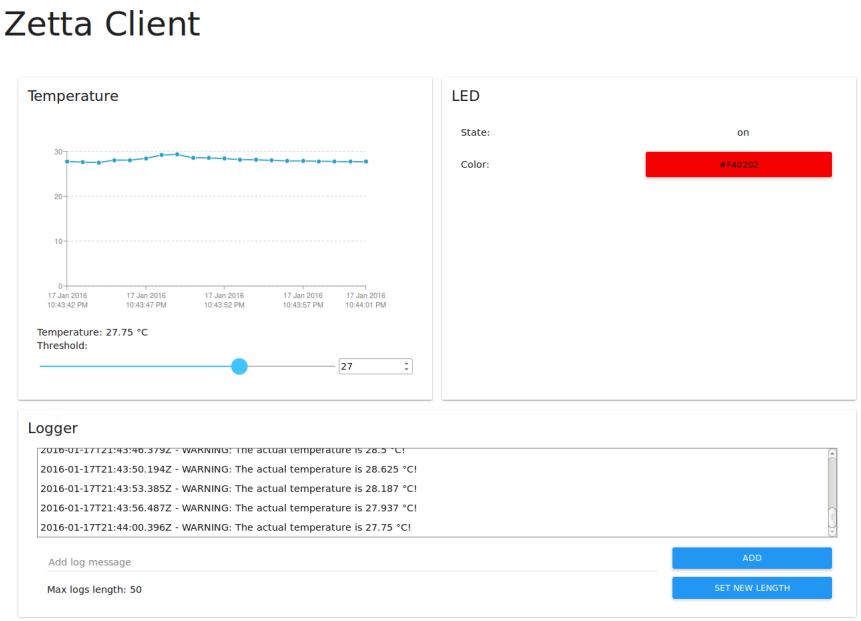

Figura 2: Screenshot dell'interfaccia del client

API. Infatti, l'utilizzo dei Web Service sembra essere uno strumento molto indicato come standard di comunicazione tra agenti diversi connessi in rete. Zetta gioca un ruolo molto importante in questo contesto (sebbene non sia stato progettato con l'idea di implementare un Multi Agent System) perchè permette di standardizzare ulteriormente l'interfaccia fornita al mondo esterno dai server (o agenti) e fornisce funzionalità ad alto livello per la ricerca (scouting) di dispositivi, per il collegamento (peering) automatico tra server e per effettuare richieste (querying) ai dispositivi.

Infine, è interessante notare come l'idea di dispositivi indipendenti nella metafora dell'Internet of Thinks (IoT) sia molto simile a quella di agenti autonomi descritta nella teoria dei Multi Agent Systems (MAS).

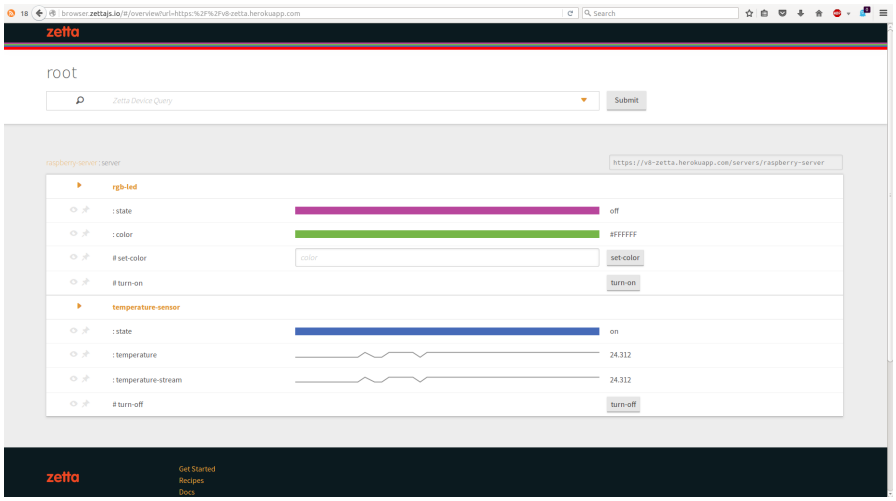

Figura 3: Screenshot di browser.zettajs.io