Relazione e guida utente infrastruttura per Web of Things

Nicola Piscaglia
nicola.piscaglia2@studio.unibo.it

Federico Pucci
federico.pucci2@studio.unibo.it

Mattia Vandi
mattia.vandi@studio.unibo.it

24 maggio 2017

Sommario

L'obiettivo dell'attività di tirocinio consiste nella progettazione e sviluppo di supporti infrastrutturali e middleware per sistemi IoT (Internet of Things) e WoT (Web of Things) che prevedono l'integrazione di dispositivi eterogenei quali Arduino e Raspberry Pi, usando REST-ful web service come architettura di riferimento e protocolli Bluetooth e Wi-Fi come tecnologie wireless.

Il nostro scopo è abilitare l'utilizzatore del middleware alla connettività con oggetti del mondo fisico, rappresentati nel nostro caso dai dispositivi Arduino, attraverso una API che interagisca con un Server REST. Quest'ultimo sarà allocato su un dispositivo Raspberry Pi che fungerà da coordinatore delle "Things" del sistema e delle richieste provenienti dall'esterno della rete. I sistemi Arduino sono visti come risorsa rappresentante la Thing ma potrebbero gestire anche più risorse come sensori e attuatori.

Il nostro obiettivo di partenza è la realizzazione di un sistema che riceva chiamate da un Client HTTP e che permetta di reperire informazioni riguardo lo stato delle risorse e di modificarne lo stato tramite chiamata REST.

Il nostro caso di studio è limitato ad un solo coordinatore Raspberry Pi dei dispositivi Arduino che vengono fisicamente contattati tramite la tecnologia Bluetooth: questa tecnologia wireless vincola il numero dei dispositivi connessi simultaneamente ad un massimo di sette per master. Per risolvere questo vincolo dettato dal Bluetooth ci prefiggiamo di mettere a disposizione due modalità di comunicazione:

- Persistente: il device Bluetooth viene connesso con un apposita chiamata ed occupa una delle sette connessioni disponibili. Da utilizzare se si necessita di efficienza in termini temporali nella comunicazione.
- Non persistente: il device Bluetooth viene connesso dinamicamente nel momento in cui si richiede una comunicazione con la Thing e la connessione viene chiusa subito dopo l'avvenuto scambio di messaggi. In questo modo non si occupano in modo persistente spazi per la connessione ed è possibile anche comunicare con più di sette dispositivi, occorre però considerare che i tempi di risposta crescono rispetto alla modalità descritta precedentemente.

L'analisi delle prestazioni del sistema verrà discussa con maggiore precisione nelle sezioni successive, per ora ci siamo limitati a descrivere logicamente il sistema.

Nella prossima sezione verranno discusse le scelte relative all'architettura delle sottoparti del sistema con maggior dettaglio.

1 Architettura del sistema

La struttura del sistema è composta dai sottosistemi:

- Master rappresentato nello specifico da Raspberry Pi 3.
- Things rappresentato da uno o più Arduino e relativi componenti.

1.1 Architettura sotto-sistema Master

Il Master gestisce le richieste provenienti dall'esterno, le indirizza alle things e restituisce al chiamante le relative risposte.

A livello software il problema è stato decomposto in tre livelli differenti che gestiscono differenti aspetti della comunicazione:

- REST-ful Server: è il layer di più alto livello. Gestisce la ricezione delle richieste provenienti da rete attraverso un server HTTP e le relative risposte ai Client. Alla ricezione di chiamate REST, le richieste vengono propagate agli strati inferiori tramite il layer sottostante detto EventBus.
- EventBus Server: rappresenta un canale di comunicazione tra gli altri due layer del sotto-sistema che si possono iscrivere ed ascoltare i messaggi sul bus per questo fa da collante tra il REST-ful Server e il CommunicationManager. Gestisce le richieste dall'alto e risposte dal basso in delle code di messaggi garantendo la corrispondenza tra request/response.
- Communication Manager: si occupa dell'indirizzamento, invio/ricezione fisica dell'informazione con i sottosistemi Things. Mantiene una lista delle Things associate disponibili e di quelle connesse nello specifico istante e controlla la comunicazione con più dispositivi offrendo la possibilità di una connessione persistente o non persistente. La comunicazione avviene utilizzando il formato JSON.

1.2 Sottosistemi Things

Rappresenta l'oggetto fisico con cui comunicare ed è gestito da un Arduino UNO.

A livello software sono stati incapsulati in apposite classi di modello i componenti passivi del sottosistema rappresentati come risorse sulla base del modello REST. Per quanto riguarda la comunicazione è stata definita una classe di gestione che controlla a polling la presenza di un messaggio in ricezione ed invia la risposta in formato JSON.

Il contenuto delle risposte è definito in modo specifico dalle risorse implementate che estendono una classe astratta AbstractResource e ne implementano i metodi concreti (GET / PUT) per soddisfare la relativa richiesta.

2 Implementazione

Sottosistema Master: è stato utilizzato il framework Vert.x per costruire un'applicazione reattiva e scalabile sulla base del modello di programmazione asincrono ad eventi con comunicazione non bloccante:

- `RESTfulServer`: implementa in Java un server REST HTTP contattabile dalla rete.
- `EventBusServer`: codificato anch'esso in Java. Permette di avviare il canale chiamato Eventbus fornito da Vert.x attraverso il quale gli altri due layer del Master comunicano. Gestisce le code di messaggi/richieste.

- `CommunicationManager`: scritto in Python. Include la libreria Pybluez per implementare la comunicazione Bluetooth con i sottosistemi Things.

Sottosistema Things: implementazione realizzata mediante linguaggio standard C/C++, usando le strutture dati incluse nella libreria AVR-STL (Standard Template Library).

La comunicazione è stata realizzata mediante il protocollo Bluetooth RFCOMM che sfrutta la seriale per inviare dati tramite protocollo Wireless Bluetooth.

Per la codifica/decodifica dei messaggi in formato JSON è stata utilizzata un apposita libreria chiamata ArduinoJson.

3 Guida utente

Abbiamo reso disponibili due implementazioni dell'infrastruttura:

- Una semplificata in cui è possibile riferirsi direttamente alla risorsa senza specificare il nome del dispositivo e la categoria alla quale appartiene.
- Ed una più complessa in cui è necessario specificare il nome del dispositivo e la categoria alla quale la risorsa appartiene.

Ai fini dell'operatività del sistema vanno avviati i seguenti componenti raggruppati per sottosistemi:

- **Sottosistema Master**: si riportano i comandi di avvio dei componenti e le istruzioni per eseguirli, ricordando che è necessario avere installato sul Raspberry Pi Vert.x 3 e Python:
  - **REST-ful Server**

    ```
    $ cd middleware/bt-connectivity/src/main/java/it/unibo/restapi
    $ vertx run RESTfulServer.java -ha
    ```

  - **Eventbus Server**

    ```
    $ cd middleware/bt-connectivity/src/main/java/it/unibo/server
    $ vertx run EventBusServer.java -ha
    ```

  - **Communication Manager**

    ```
    $ cd middleware/bt-connectivity/src/main/python
    $ python communicationmanager_v2.py
    ```

- Sottosistemi Things: è sufficiente caricare sul microcontrollore lo Sketch Arduino "testbluetooth" presente nella cartella del progetto `middleware/bt-connectivity/src/main/arduino/testbluetooth`.

Dopo aver caricato tutti i componenti descritti in precedenza è possibile contattare il RESTful Server all'indirizzo associato al Raspberry Pi sulla porta 8080 attraverso un qualsiasi REST Client (es: Browser Web, Advanced Rest Client, Postman, ecc). Lista delle URI e dei metodi disponibili:

- `/api/inquiry`
  - GET: restituisce un elenco delle risorse disponibili non connesse in maniera persistenti. Parametri:
    - `duration`: numero di secondi impiegati per la ricerca (opzionale, default 8 secondi)

- `/api/resources`
  - GET: restituisce una lista degli identificativi delle risorse connesse in maniera persistente.
  - POST: tenta di connettere la risorsa indicata. Parametri:
    - `resourceId`: identificativo della risorsa che si vuole connettere
- `/api/resources/:resourceId`
  - GET: restituisce lo stato della risorsa indicata. Il corpo della risposta può essere definito dallo sviluppatore durante l'implementazione delle singola risorsa.
  - PUT / PATCH: modifica lo stato della risorsa indicata. I parametri possono essere definiti dallo sviluppatore durante l'implementazione delle singole risorse.
  - DELETE: elimina la connessione persistente precedentemente creata o restituisce errore se non è stata creata in precedenza.

4 Risultati ottenuti

Il sistema realizzato permette di leggere e modificare risorse attraverso chiamate REST con un buon grado di affidabilità nella comunicazione anche se non sempre in maniera tempestiva.

I tempi di risposta alle chiamate effettuate da noi rilevati, si aggirano in media sui 2,5 secondi utilizzando una connessione non persistente e sui 0,5 secondi utilizzando una connessione persistente.

La maggior parte del ritardo nella comunicazione è dettata dal limite fisico del Bluetooth utilizzato sia per la procedura di connessione (circa 2 secondi) sia per l'invio/ricezione di dati (dai 100 ai 400 ms circa): infatti provando ad effettuare richieste che non necessitano di comunicazione BT (inoltrate fino allo strato del Communication Manager), si ottengono tempi di risposta sull'ordine dei 90 ms.

Mentre i tempi di risposta ottenuti con connessione persistente sono accettabili, la comunicazione con connessione dinamica risulta eccessivamente lenta. Quest'ultima però ha il vantaggio di scollegare automaticamente il dispositivo al termine della comunicazione il che influisce positivamente sul numero di connessioni simultanee disponibili (massimo 7).

Per questo se si necessita di prestazioni in termini temporali è indispensabile utilizzare connessioni persistenti mentre se non si hanno vincoli temporali stringenti ed un numero di dispositivi maggiore di sette è possibile fare ricorso a connessioni non persistenti.

5 Possibili sviluppi futuri

Il naturale sviluppo di questo progetto è rappresentato dall'integrazione dell'interazione del mondo fisico con il mondo digitale.

Sarebbe interessante sviluppare la parte di comunicazione che viene scatenata dagli effetti sul mondo fisico e propaga informazione attraverso i vari sottosistemi fino ad arrivare in maniera digitale all'utente.

Un semplice esempio con cui partire potrebbe essere l'invio dello stato del bottone su una pagina web alla pressione dello stesso (e non su richiesta GET dell'utente).

L'implementazione potrebbe essere realizzata a livello di Master attraverso l'utilizzo di Web Socket, mentre lato Things si potrebbe associare un interrupt al bottone oppure utilizzare una strategia a polling.

Un ulteriore sviluppo del progetto potrebbe riguardare l'inserimento di un pool di ThreadExecutor sulla parte del Communication Manager in Python al posto di allocare thread per ogni

nuova connessione, al fine di ottimizzare il consumo di risorse in termini di memoria all'aumentare delle connessioni da gestire.

6 Riferimenti

Lista delle varie tecnologie/framework utilizzati per lo sviluppo dell'infrastruttura:

1. Vert.x Core
2. Vert.x Web
3. Vert.x TCP Event bus Bridge
4. Python Vert.x TCP Event bus Client
5. Pybluez
6. Standard Template Library for AVR
7. ArduinoJson