Interfaccia client-side JavaScript per TuSoW

Luca Bracchi
luca.bracchi3@studio.unibo.it

Aprile 2021

Contents

1 Introduzione

Linda è un modello di coordinazione e comunicazione tra processi distribuiti che operano su oggetti mantenuti in un tuple space. L'idea di base è quella di fornire un modello basato su un set di primitive di coordinazione ristretto, ma sufficiente per progettare un sistema distribuito.

La comunicazione tra gli agenti nel modello Linda è basata su tuple, collezioni ordinate di informazioni. Le tuple sono mantenute dal tuple space e gli agenti comunicano scrivendo, leggendo o prendendo tuple dal tuple space. I modelli di coordinazione tuple-based sono tra i più studiati per via della loro espressività, eleganza e flessibilità.

Le operazioni richieste dal modello Linda sono:

- `out`(write) permette a un agente di inserire una tupla nel tuple space
- `rd`(read) permette a un agente di leggere una tupla dal tuple space senza consumarla; se la tupla non è disponibile, la richiesta viene mantenuta in sospeso fino all'inserimento della tupla
- `in`(take) simile a `rd`, ma la tupla designata viene rimossa dal tuple space; se la tupla non è disponibile, la richiesta viene mantenuta in sospeso fino all'inserimento della tupla

I dati presenti nel tuple space vengono selezionati utilizzando i template, che mantengono una conoscenza totale o parziale del dato che si vuole recuperare.

Tuple Space over the Web (TuSoW) fornisce un'implementazione leggera, flessibile, modulare e altamente interoperabile di un tuple space basato sul modello Linda e adatto per il coordinamento di agenti software in ambienti che adottano il paradigma dell'edge computing, ovvero dove la computazione avviene il più vicino possibile a dove vengono richiesti i dati.

2 Obiettivi

In questa sezione sono elencati gli obiettivi del progetto:

- Studiare il sistema TuSoW: principi su cui si basa, architettura e in particolare i servizi REST che offre.
- Implementare una libreria JavaScript client-side che permetta di interfacciarsi col server TuSoW: l'interfaccia ha lo scopo di permettere al client di interagire con il tuple space attraverso chiamate HTTP.
- Testare l'interfaccia implementata

La libreria deve supportare le stesse operazioni offerte dal server TuSoW, ovvero:

- `count()`: restituisce il numero di tuple contenute nel tuple space

- `out(T)`: inserisce la tupla `T` nel tuple space
- `get()`: legge tutte le tuple del tuple space senza consumarle
- `in(TT)`: consuma e restituisce una tupla dal tuple space che faccia match con il template `TT`; se il match non da risultati l'esecuzione viene bloccata finchè non viene aggiunta al tuple space una tupla che faccia match con `TT`
- `inp(TT)`: si comporta come `in(TT)`, ma se nessuna tupla fa match viene restituito un messaggio di errore, senza bloccare la computazione
- `in_all(TT)`: si comporta come `in(TT)` ma restituisce la collezione di tutte le tuple che fanno match con il template `TT` e se nessuna tupla fa match viene restituita una collezione vuota
- `rd(TT)`: si comporta come `in(TT)` ma senza consumare la tupla restituita
- `rdp(TT)`: si comporta come `inp(TT)` ma senza consumare la tupla restituita
- `rd_all(TT)`: si comporta come `in_all(TT)` ma senza consumare le tuple restituite
- `no(TT)`: controlla se ci sono tuple nel tuple space che fanno match con il template `TT`; se non ci sono l'operazione ha esito positivo, altrimenti l'esecuzione viene bloccata finchè tali tuple si trovano nel tuple space
- `nop(TT)`: si comporta com `no(TT)` ma, nel caso in cui ci sia una tupla che fa match con `TT`, questa viene restituita senza che l'esecuzione venga bloccata

3 Piano di lavoro

In questa sezione sono elencate le fasi previste dal piano di lavoro relativo al progetto:

- Analisi e comprensione del sistema TuSoW descritto nell'articolo [CROM19]
- Progettazione e sviluppo dell'interfaccia: il sistema supporterà tuple space testuali e logici
- Test del prodotto utilizzando il framework Mocha e l'assertion library Chai
- Valutazione dei risultati raggiunti

Lo studio del sistema TuSoW comprenderà il reverse engineering delle API offerte con conseguente stesura della relativa specifica Swagger. Vista la dimensione del progetto, verrà usato NPM come tool di build automation. Per quel che riguarda il deploy verrà impiegato lo strumento di CI offerto da Gitlab per automatizzare le fasi di build e pubblicazione del modulo su npmjs.

4 Analisi

L'utente che usa il modulo sviluppato desidera interagire con un tuple space per eseguire operazioni sulle informazioni che contiene, informazioni codificate come tuple. Il tuple space è mantenuto da un server, in questo caso un server TuSoW, che permette ai client di connettervisi e di utilizzare le operazioni fornite per ottenere le informazioni desiderate.

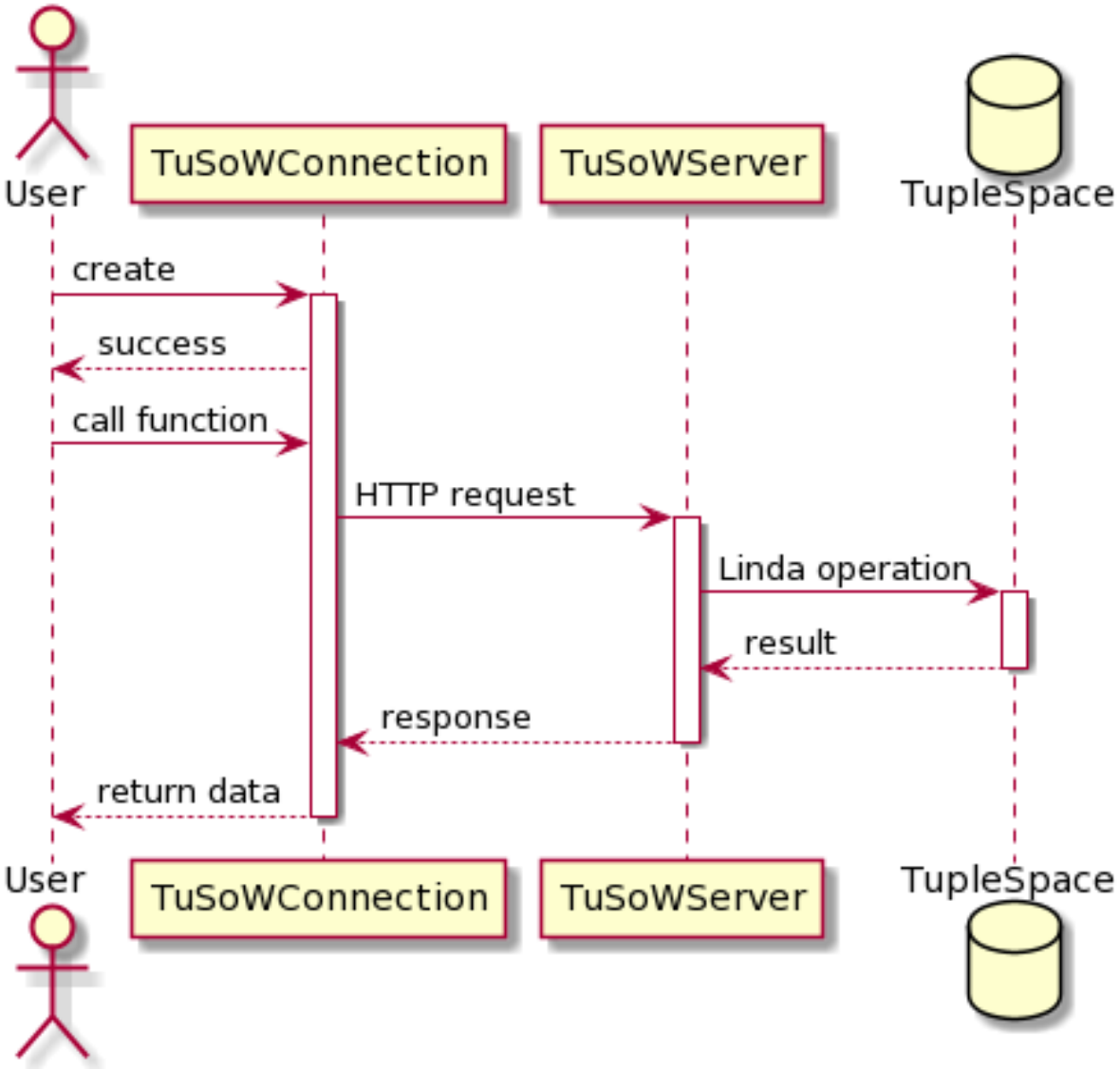

Figure 1: Rappresentazione di sequenza

Lo scopo del modulo è quello di porsi tra l'utente e il server e di rendere possibile lo scambio

di richieste e relative risposte in maniera semplice e rispettando il comportamento delle operazioni supportate, soprattutto per quel che riguarda il fatto che l'operazione sia sincrona o asincrona e che vada quindi a interrompere l'esecuzione oppure no. Il diagramma di sequenza illustrato nella figura 1 mostra la serie di operazioni che idealmente portano l'utente a ottenere le informazioni desiderate dal tuple space, prima istanziando una connessione con il server TuSoW e successivamente utilizzando le API offerte dal costrutto che implementa la connessione per far sì che il server esegua le operazioni volute sul tuple space e restituisca il risultato di tali operazioni.

La logica del tuple space e delle operazioni effettuabili su questo è delegata interamente al server, il modulo `tusow` si occupa principalmente di formattare nella maniera adeguata le richieste effettuate al server, di eseguire tali richieste e di restituire in un formato adeguato le risposte ricevute. Il server si occupa della serializzazione e deserializzazione dei body delle richieste HTTP da JSON a oggetti e viceversa, ma lato client è necessario convertire tuple e template in formato JSON prima di inserirli nel body delle richieste. Questo punto verrà trattato nella sezione dedicata all'implementazione.

4.1 Tuple space

Il modulo deve supportare tuple space testuali e logici. Nei primi le informazioni sono codificate come tuple testuali, ovvero stringhe, e usano come template le espressioni regolari; i secondi usano i termini Prolog sia come tuple che come template. I template dei tuple space logici devono essere termini non-ground, ovvero termini che non contengono variabili.

I termini Prolog sono composti da un atomo, chiamato 'functor', e un determinato numero di argomenti(anche nessuno) che sono a loro volta termini. I termini sono scritti come un functor seguito dalla lista degli argomenti associati tra parentesi e separati da virgole. Il listing 1 mostra un esempio di termine seguito dalla sua presentazione come oggetto, che è il formato nel quale vengono passati i termini come argomenti delle funzioni del modulo descritto nella prossima sezione.

```
1  // a Prolog term
2  message(sender, receivers(A,B), content('Hello!'))
3  // an object-converted Prolog term
4  {
5    fun:"message",
6    args: [
7      "sender",
8      { fun:"receivers", args:["A","B"]},
9      { fun:"content", args:["Hello!"]}
10   ]
11 }
```

Listing 1: Esempio di termine Prolog e oggetto JavaScript associato

4.2 Reverse engineering

Prima di tutto è stato studiato il sistema TuSoW per individuare gli endpoint da utilizzare nel modulo da sviluppare e le operazioni associate a questi. Sono state analizzate anche le classi componenti del backend che sarebbero tornate utili durante le fasi successive, in particolare quelle che si occupano della serializzazione e deserializzazione di tuple, template e match tra template e tuple.

TuSoW è stato testato eseguendo un'istanza del server in locale e usando l'estensione REST Client di Visual Studio Code per simulare chiamate REST al server, vederne i risultati e tenere traccia di tali chiamate in un file `.http`. Il listing 2 mostra un esempio di chiamata HTTP eseguibile attraverso lo strumento REST Client per scrivere una tupla nel tuple space testuale `ts` e per leggere poi tutte le tuple che fanno match con il template passato nel body, tra le quali troviamo la tupla inserita con la prima chiamata.

```
### write a tuple
POST http://localhost:8080/tusow/v1/tuple-spaces/textual/ts
content-type: application/json
accept: application/json
{
    "tuple": "fun:message;args:john,smith"
}
### read a tuple
GET http://localhost:8080/tusow/v1/tuple-spaces/textual/ts
content-type: application/json
accept: application/json
{
    "template": "fun:(?<name>.*);args:(?<sender>.*),(?<receiver>.*)"
}
```

Listing 2: Esempio di chiamate HTTP al server TuSoW per leggere e scrivere tupla

4.3 Specifica Swagger

Il reverse engineering delle API offerte da TuSoW ha portato alla stesura di una specifica OpenAPI, reperibile su SwaggerHub [Bra], che raccoglie gli endpoint offerti dal server e le operazioni effettuabili in base al metodo HTTP utilizzato.

Ogni metodo HTTP supporta un insieme di operazioni sul tuple space; in particolare:

- HEAD: `count()`
- GET: `get()`, `rd(TT)`, `rdp(TT)`, `rd_all(TT)`, `no(TT)`, `nop(TT)`
- POST: `out()`
- DELETE: `in(TT)`, `inp(TT)`, `in_all(TT)`

La specifica comprende i servizi REST offerti per i tuple space testuali e per quelli logici.

Sempre nella specifica si trovano gli schemi relativi a tuple, liste di tuple, tuple match e template per entrambi i tipi di tuple space.

5 Implementazione

Le funzioni offerte dal modulo tusow accettano strighe, ma anche oggetti generici come parametri di input per quel che riguarda tuple e template, permettendo di interfacciarsi con qualunque tipo di tuple space e non solo quelli testuali.

L'implementazione dell'interfaccia relativa ai tuple space logici si compone infatti di funzioni che accettano in input oggetti generici, questo permette al modulo di proporsi come soluzione client side anche per tuple space non trattati in questo documento, senza la necessità di sviluppi addizionali eccessivi.

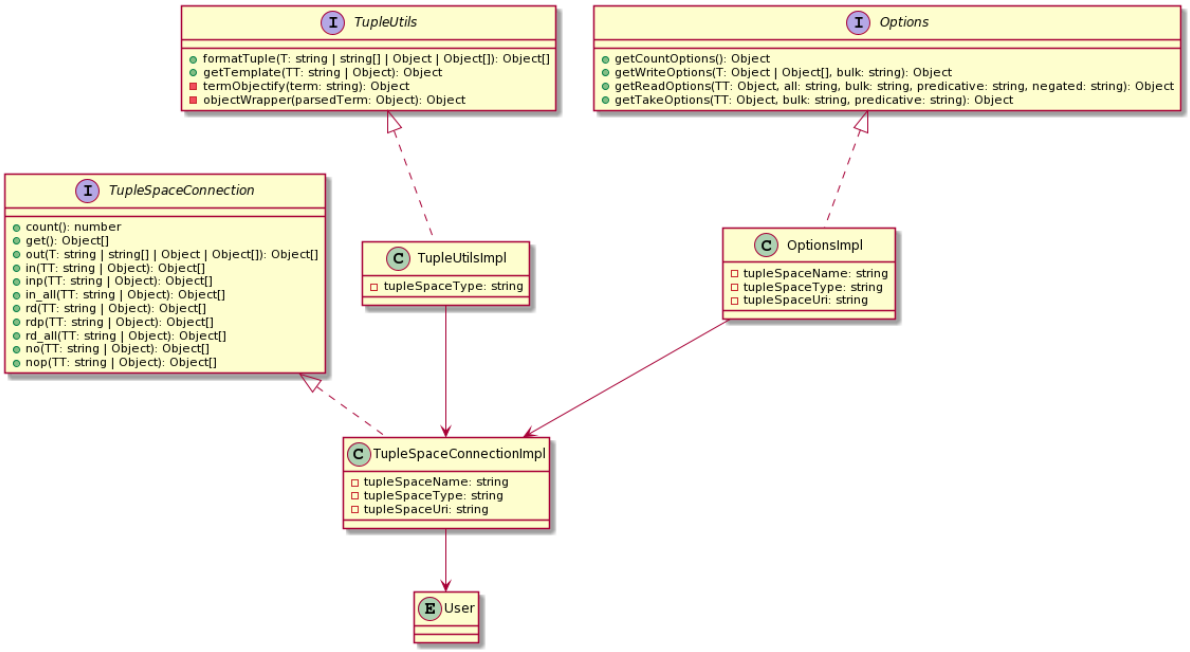

Figure 2: Diagramma delle classi del modulo `tusow`

5.1 Strumenti utilizzati

Sono stati utilizzato i seguenti strumenti per l'implementazione:

- Axios come client HTTP
- tuProlog per quel che riguarda serializzazione e deserializzazione delle tuple logiche

- TypeScript come linguaggio del modulo

5.2 Axios

Il modulo npm `axios` è stato usato come HTTP client per eseguire le richieste verso il server. Oltre ad essere utilizzabile sia da node.js che da browser, `axios` serializza automaticamente gli oggetti JavaScript in formato JSON e si occupa anche della deserializzazione dei JSON nelle response in oggetti JavaScript, in modo tale da evitare errori dovuti alla non serializzazione/deserializzazione dei body di request e response HTTP.

Per quel che riguarda le chiamate alle funzioni del module che si occupano di eseguire chiamate HTTP verso il server è stato utilizzato il costrutto async/await di JavaScript per evitare di avere a che fare con funzioni di callback, che solitamente rendono il codice più difficile da comprendere. Per lavorare con le Promise nel contesto JavaScript è possibile usare le keyword async e await: la prima, posta prima della funzione che si sta definendo, indica che questa ritornerà sempre una Promise, quindi i valori restituiti saranno wrappati in una Promise. La keyword await viene usata all'interno di una funzione definita con la keyword async e fa sì che JavaScript aspetti che la Promise venga risolta e che ritorni un risultato prima di riprendere l'esecuzione.

5.3 tuProlog

Per quel che riguarda la conversione da stringa a tupla logica sono stati utilizzati i moduli serialize-core e parsing-core del framework tuProlog [tuP] compilato in JavaScript. Prima di tutto sono stati studiati codice e documentazione per determinare le funzionalità offerte dai vari moduli.

In seguito è stata implementata una funzione che prende in input una stringa che rappresenta un termine, la converte in un oggetto JavaScript e infine si appoggia a una funzione ricorsiva per fare il wrap dell'oggetto in un altro formattato in maniera tale da essere accettato dalle API del server TuSoW come input. La funzione viene usata per convertire tuple e template logici, essendo entrambi termini Prolog.

Per quel che riguarda la trasformazione in stringa delle tuple logiche restituite dal server TuSoW, è stata implementata una funzione ricorsiva che copre tale mansione.

5.4 TypeScript

Per sviluppare il modulo è stato usato TypeScript, superset di JavaScript, per i vantaggi e le funzionalità che offre, come la definizione di classi e interfacce o la possibilità di specificare i tipi dei parametri e i tipi di ritorno delle funzioni. TypeScript supporta inoltre il type-checking statico, cioè i controlli sui tipi avvengono a compile time, evitando errori difficilmente analizzabili a runtime.

In fase di sviluppo è stato usato il watcher per tener controllati i file `.ts` contenuti in una cartella specificata nel file di configurazione di TypeScript; in questo modo quando uno di questi file viene modificato, viene immediatamente compilato e il file JavaScript ottenuto viene messo in una cartella di output, sempre specificata nel `tsconfig.ts`.

6 Testing

I test sul sistema sono stati svolti usando [Moc] come testing framework. Mocha supporta le Promises out-of-the-box, semplificando il testing di funzioni asincrone, e permette l'uso di diverse assertion library; per i test del progetto è stato impiegato [Cha].

Mocha è un test framework per Node.js e browser che permette di descrivere le feature implementate tramite la funzione `describe`. Nella callback di `describe` si possono segmentare i test inserendo ulteriori chiamate di `describe`, oppure introdurre una funzione `it`, che ha un'interfaccia simile a `describe`, come mostrato nel listing 3. Nella funzione callback di `in` si possono usare i costrutti offerti dall'assertion library scelta per testare concretamente le feature desiderate.

```
1  // describe interface
2  describe: (description: string, callback: () => void) => void;
3  //it interface
4  it: (description: string, callback: () => void) => void;
```

Listing 3: Interfacce delle funzioni describe e it di Mocha

Le diverse interfacce di Chai offrono la possibilità di scrivere test utilizzando un linguaggio espressivo e facilmente leggibile nel caso di Should e Expect, oppure un approccio più tradizionale se si sceglie Assert. Il listing 4 mostra a titolo esemplificativo come testare se l'API `count` del modulo `tusow` ritorna effettivamente un valore numerico oppure no.

```
1  // Expect
2  var expect = chai.expect();
3  expect(tupleSpaceConnection.count()).to.be.a("number");
4  // Should
5  chai.should();
6  tupleSpaceConnection.count().should.be.a("number");
7  // Assert
8  var assert = chai.assert;
9  assert.typeOf(tupleSpaceConnection.count(), "number");
```

Listing 4: esempi di utilizzo delle interfacce Expect Should e Assert di Chai

Per quanto riguarda i test del modulo, situati nella cartella "test", sono stati separati quelli riguardanti i tuple space testuali, raccolti in un file, da quelli riguardanti quelli logici, raccolti in un altro. Ogni file contiene i test volti a controllare che la semantica delle operazioni sui tuple space si rispettata, inizializzando delle connessioni con dei tuple space di test e richiamando le API del modulo, per poi controllare se la risposta ricevuta corrisponde a quella attesa.

Mocha esegue i test in maniera sequenziale, questo rende difficile testare la proprietà bloccante delle operazioni sincrone, visto che il flusso di esecuzione non può essere in attesa di un evento e far avvenire contemporaneamente l'evento stesso. Per questo motivo è stato aggiunto un file "util.js" dove vengono eseguite le operazioni necessarie per sbloccare le operazioni asincrone, quali `in(TT)`, `rd(TT)` e `no(TT)` e utilizzando il parametro `--parallel` di Mocha perché gli script dei test vengano eseguiti in parallelo, in questo modo quando lo script relativo ai tuple space logici si trova bloccato in un'operazione di read sincrona dovrà solo aspettare che lo script di supporto inserisca nel tuple space la tupla necessaria per terminare l'operazione e riprendere l'esecuzione. Il test del modulo senza lo script di supporto ha come conseguenza il fallimento dei controlli sulle proprietà delle operazioni sincrone, in quanto rimarranno in attesa di un evento per un tempo pari a quello del timeout impostato automaticamente da Mocha per le chiamate REST, che corrisponde a 2000ms.

7 Deployment

Il progetto è stato deployato su [npm] come package npm. Per usare la libreria in un progetto esterno è sufficente installare il package `tusow` e importarlo con la funzione `require('tusow')`. Sono stati utilizzati gli strumenti di CI offerti da GitLab per automatizzare il processo di compilazione dei file TypeScript e per pubblicare la nuova versione del modulo appena compilata su npmJs.

Per quel che riguarda la pubblicazione automatica del modulo è importante aggiornare la versione prima di fare il push dei commit: l'aggiornamento può essere fatto direttamente nel `package.json` oppure tramite gli appositi comandi `npm version patch`, `npm version major` o `npm version minor`.

I test non sono integrati con il processo di CI, ma per testare il modulo è sufficiente istallarlo, compilare i file Typescript ed eseguire `npm run test`, comando definito nel `package.json`.

8 Esempio d'uso

Nella presente sezione verrà mostrato un esempio di utilizzo pratico del modulo `tusow`. Innanzitutto è necessario istallare il suddetto pacchetto e le relative dipendenze con il comando `npm install tusow` e importarlo come mostrato nel listing 5.

```
1  // import in JavaScript
2  const tusow = require('tusow');
3  // import in TypeScript
4  import { TupleSpaceConnectionImpl } from "tusow";
```

Listing 5: Import del modulo ́tusowín JS e TS

Successivamente bisogna istanziare una connessione con il server TuSoW, passando come parametri il nome del tuple space a cui collegarsi, il tipo "textual" se si tratta di un tuple space

testuale o "logic" se è invece logico e infine l'uri a cui risponde il tuple space. Sarà poi possibile utilizzare le funzioni fornite dal modulo per interfacciarsi col tuple space. Il listing **??** mostra un esempio di istanziamento della connessione e di operazione di scrittura e successiva lettura di una tupla testuale.

```
1  // istanzio una connessione
2  const connection = new TupleSpaceConnectionImpl("ts", "textual", "http://
       localhost:8080/tusow/v1/tuple-spaces");
3  // inserisco una tupla nel tuple space
4  connection.out("name:tuple;function:example");
5  // leggo la tupla dal tuple space con una read non bloccante
6  connection.rd(".*name:tuple.*");
```

Listing 6: Esempio d'uso del modulo `tusow` per tuple space testuali

Il listing 7 mostra invece come eseguire le stessa operazioni del listing precedente in un tuple space logico. Cambiando i parametri di input del costruttore della connessione e delle operazioni che questa fornisce è possibile interfacciarsi con qualsiasi altro tipo di tuple space.

```
1  // istanzio una connessione
2  const connection = new TupleSpaceConnectionImpl("ts", "logic", "http://
       localhost:8080/tusow/v1/tuple-spaces");
3  // inserisco una tupla nel tuple space
4  connection.out( "message(sender,receiver)" );
5  // leggo la tupla dal tuple space con una read non bloccante
6  connection.rd( "message(sender,receiver)" );
```

Listing 7: Esempio d'uso del modulo `tusow` per tuple space logici

9 Conclusioni

Si è studiato il sistema TuSoW, gli endpoint che offre e le classi che lo compongono, raccogliendo i dati ottenuti in una specifica Swagger. In seguito è stato sviluppato il modulo `tusow` che permette di interfacciarsi in maniera semplice e intuitiva con il server TuSoW per operare su tuple space testuali e logici. Una serie di test sono stati sviluppati usando il framework Mocha per verificare la correttezza della semantica delle operazioni offerte dal package. Infine è stato integrato un semplice script di CI che permette, ad ogni commit, di fare la build e il deploy del modulo in maniera automatica.

In futuro potrebbe essere utile integrare i test alla pipeline di Gitlab, lanciando un'immagine Docker su cui viene eseguito il server TuSoW e che esponga l'endpoint necessario per testare le varie funzioni offerte dal package.

References

[Bra] Luca Bracchi. Specifica swagger per tusow. URL: `https://app.swaggerhub.com/apis/lbracchi/tusow-specs/1.0.0`.

[Cha] Chai. URL: `https://www.chaijs.com`.

[CROM19] Giovanni Ciatto, Lorenzo Rizzato, Andrea Omicini, and Stefano Mariani. Tusow: Tuple spaces for edge computing. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6. IEEE, 2019.

[Moc] Mocha. URL: `https://mochajs.org/`.

[npm] npmjs. URL: `https://www.npmjs.com`.

[tuP] tuprolog. URL: `https://www.npmjs.com/org/tuprolog`.