

speech2owl

Tema d'anno di Linguaggi e Tecnologie Web

Vito Domenico Tagliente

Matricola 567130

Sorgente disponibile su: <https://github.com/vitodtagliente/speech2owl>

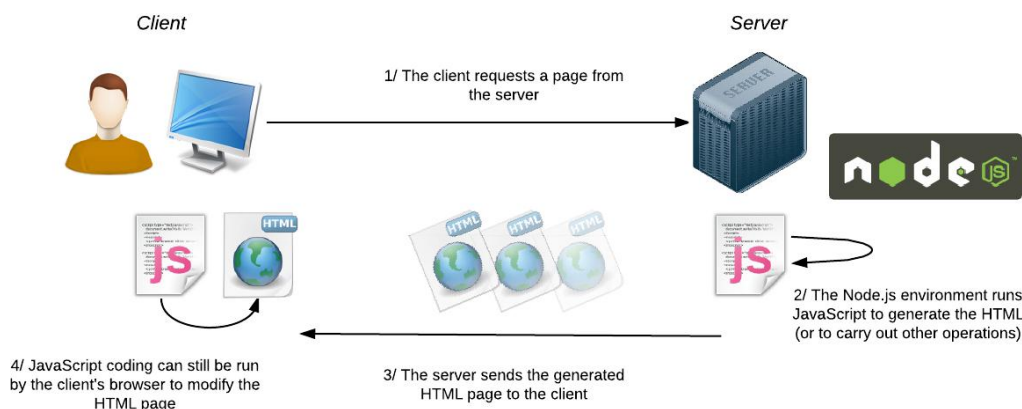
1. Introduzione

Scopo del progetto è stato quello di sviluppare una applicazione web-based in grado di creare una annotazione OWL (rispetto ad una ontologia di riferimento) a partire da un messaggio vocale dell'utente.

Per la realizzazione di tale applicazione è stata adottata una architettura client-server basata su tecnologia Node.js.

Node.js è una piattaforma realizzata su V8, il motore Javascript di Chrome, che permette di realizzare applicazioni web veloci e scalabili. Node usa un modello ad eventi e un sistema di I/O non bloccante che lo rende leggero ed efficiente, perfetto per applicazioni real-time che elaborano dati in modo intensivo e che può essere distribuito su più sistemi.

Prima considerazione: fin qua nessuno ha parlato di server, ed in effetti è così! Di solito si liquida Node dicendo che è un "server in javascript", ma la cosa è in parte vera: Node è più in generale "javascript lato server", ovvero è un Motore Javascript, che, tra le altre cose, è capace di tirar su un server HTTP o TCP in pochissime righe di codice! Il server quindi non è qualcosa di scorrelato dall'applicativo web che vogliamo realizzare, ma è una caratteristica dell'applicativo stesso!



Chiaramente per la realizzazione della parte client dell'applicazione si sono adottate le tecnologie standard per interfacce web, quali HTML5 + CSS + Javascript.



Perchè Node.js? Perchè una architettura client-server?

L'adozione di tale scelta è stata indirizzata dalla disponibilità di numerosi pacchetti, sviluppati dalla community, per ambiente node. Parliamo di pacchetti che, in ambiente browser, non esisterebbero a causa dei vincoli e dei limiti vigenti. Tutto ciò è stato inoltre guidato da un'ottica di migioria e sviluppo futuro dell'applicazione, in quanto risulterà semplice aggiungere nuove funzionalità, anche complesse, semplicemente installando dei nuovi pacchetti ed aggiornando le API REST dell'applicazione stessa.

2. Il Server

Per quanto riguarda l'implementazione del lato server sono stati utilizzati i seguenti pacchetti:

1. [express](#), un framework semplice per la realizzazione di un server HTTP in ambiente node. Questo permette di definire API REST dei servizi messi a disposizione in tutta semplicità ed in poche righe di codice.
2. [colors](#) è un semplice framework per la colorazione dei log sulla console del server.
3. [pos](#) è il cuore del progetto. Parliamo di un pacchetto di NLP che permette di eseguire lo speech-tagging, una tecnica che consiste nell'esaminare del testo e associare dei tag ad ogni token costituente. Tali tag andranno a specificare se un determinato token sia un nome, verbo, etc...

Nell'implementazione corrente del server, questo si occupa di gestire due sole richieste HTTP di tipo GET. Specificate le routes:

1. [/](#): ritorna la vista dell'applicazione contenuta nel file [index.html](#).
2. [/nlp](#): accetta del testo come parametro e ritorna una serializzazione json del risultato del Speech-Tagging eseguito dal modulo pos, prima menzionato.

Per avviare il server occorre eseguire il seguente comando da shell:

```
>> npm start
```

```
menick@menick-X555LPB: ~/Scrivania/speech2owl
menick@menick-X555LPB:~/Scrivania/speech2owl$ npm start

> speech2owl@1.0.0 start /home/menick/Scrivania/speech2owl
> node main.js

speech2owl is listening on port 8000!
```

A questo punto, come già specificato, puntando, con il browser, all'indirizzo `http://localhost:8000`, il server risponderà con l'interfaccia grafica dell'applicazione (ovvero con il contenuto del file `index.html`).

Invece, puntando all'indirizzo `http://localhost:8000/nlp?text=sometext`, si otterrà la serializzazione json del risultato dell'operazione di nlp sul testo specificato.

Richiesta: <http://localhost:8000/nlp?text=hello%20world>

Risultato:

```
[{
  "sentence": "hello world",
  "terms": [
    {"token": "hello", "tag": "UH"},
    {"token": "world", "tag": "NN"}
  ]
}]
```

Ogni tag ha un significato specifico:

NN Noun, sing. or mass	dog
NNP Proper noun, sing.	Edinburgh
NNPS Proper noun, plural	Smiths
NNS Noun, plural	dogs

La lista completa è riportata nel file `tags.txt`, presente nella root directory dell'applicazione.

```
menick@menick-X555LPB: ~/Scrivania/speech2owl
menick@menick-X555LPB:~/Scrivania/speech2owl$ npm start

> speech2owl@1.0.0 start /home/menick/Scrivania/speech2owl
> node main.js

speech2owl is listening on port 8000!

NLP::Text = hello world

hello world

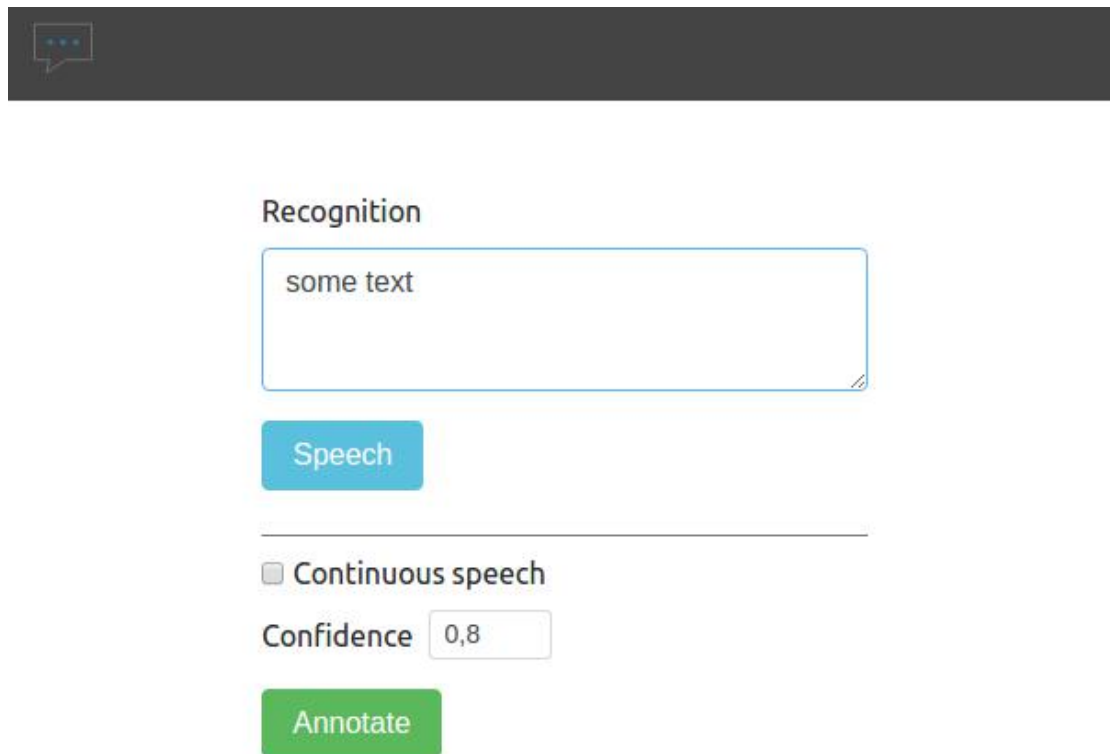
hello /UH
world /NN
```

3. Il Front-end

Per quanto riguarda il lato client dell'applicazione, l'interfaccia grafica, come già detto, è stata sviluppata in HTML5+CSS+Javascript.

Per l'impaginazione e la rappresentazione degli elementi grafici è stato utilizzato [Bootstrap](#), in particolare la versione 4 alpha. Bootstrap è uno dei framework più utilizzati per lo sviluppo di front-end, in particolare permette uno sviluppo semplice e veloce, dovuto alla buona documentazione ed alla numerosa quantità di esempi presenti sul web.

L'interfaccia grafica è stata pensata molto semplice, presentandosi come segue:



Recognition

some text

Speech

☐ Continuous speech

Confidence 0,8

Annotate

Come si nota, questa presenta una textarea per l'inserimento del testo su cui eseguire le operazioni di annotazioni e/o l'autoinserimento dovuto al modulo di riconoscimento vocale. Una volta inserito il testo è possibile annotarlo cliccando sull'apposito pulsante, potendo specificare anche il fattore di confidenza.

4. Web Speech API

L'operazione di conversione del parlato in testo è stata implementata utilizzando le API fornite da Google. Tali API sono disponibili soltanto per browser Chrome, anche se esiste una implementazione equivalente anche per Mozilla. Per sfruttare tali potenzialità, è stato sviluppato un modulo apposito per un utilizzo semplificato. In particolare, per la realizzazione di questo modulo sono stati applicati due design pattern:

1. Adapter: in quanto il modulo non fa altro che ridefinire l'interfaccia delle API di Google
2. Singleton: per permettere l'accessibilità semplificata all'istanza dell'oggetto (speech2owl.Speech.singleton).

Per utilizzare il modulo realizzato, occorre includerlo:

```
<script src="speech2owl/speech.js"></script>
```

Ed instanziarlo:

```
var speech = new speech2owl.Speech(  
    'speechTextarea', 'speechButton', true  
);  
speech.bindCheckbox( 'speechContinuous' );
```

- ✓ speechTextarea è l'id della textarea in cui verrà salvato il testo riconosciuto.
- ✓ speechButton è, rispettivamente, l'id del pulsante abilitato all'attivazione dell'ascolto.
- ✓ Il terzo argomento è un booleano che serve ad abilitare o disabilitare i log del modulo.
- ✓ speechContinuous è l'id della checkbox, facoltativa, su cui andare a impostare la modalità di ascolto del modulo, che può essere continua o meno.

5. Parsing di Ontologie

Affinchè sia possibile eseguire un match del testo riconosciuto dal modulo di Speech con una ontologia di riferimento, è stato necessario ricercare una libreria per eseguirne il parsing. A tal proposito, la scelta è ricaduta sulla libreria "Simple javascript RDF Parser and query thingy", disponibile all'indirizzo <http://www.jibbering.com/rdf-parser/>.

A questo punto, come fatto per lo Speech Recognition, ho implementato un modulo per la gestione dell'ontologia, utilizzando i pattern Adapter e Singleton.

Per utilizzare il modulo, occorre includerlo:

```
<script src="speech2owl/rdf.js"></script>
```

Per eseguire il parsing di un file rdf/owl in serializzazione xml occorre:

```
var rdf = new speech2owl.RDF( 'assets/pizza.owl', false );
rdf.onload = function(){
    // Loaded, do something
}
```

A questo punto è possibile utilizzare le API che sono state realizzate per semplificare le operazioni di ricerca e/o matching all'interno dell'ontologia.

Per prima cosa occorre definire i namespace, se necessari, per esempio:

```
Var RDFS = new
speech2owl.RDF.Namespace( 'http://www.w3.org/2000/01/rdf-schema#' );
```

Per esempio per ottenere tutte le triple ?subject ?predicate ?object

```
var results = rdf.any(subject, predicate, object);
```

Combinando le due cose, con la seguente istruzione:

```
var results = this.ontology.any(null, RDFS.get('label'), null);
```

Si ottiene il comportamento equivalente all'esecuzione della query SPARQL:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT ?subject ?object
```

```
WHERE { ?subject rdfs:label ?object }
```

A questo punto è possibile definire anche un match tra le triple ottenute dalla ricerca tramite metodo 'any' ed una generica stringa, potendo definire un fattore di confidenza, compreso tra 0 e 1.

```
var match = rdf.any(subject, predicate, object).match( 'text', confidence );
```

Se non specificato, il fattore di confidenza viene valorizzato a 0,8.

Il match ottenuto sarà un array di triple, a questo punto è possibile ottenere il match migliore, semplicemente richiamando il metodo 'best'.

```
var best = rdf.any(subject, predicate, object).match( 'text' ).best();
```

6. Distanza Jaro-Winkler

Nel precedente capitolo si è parlato di matching tra stringhe. Tale operazione è stata possibile utilizzando l'algoritmo di distanza Jaro-Winkler, che non è altro che un algoritmo che valuta la somiglianza tra due stringhe fornendo un valore numerico compreso tra 0 e 1. Chiaramente, più questo valore è alto, più le due stringhe confrontate sono simili.

Per il progetto corrente è stata utilizzata l'implementazione disponibile all'indirizzo <https://github.com/thsig/jaro-winkler-JS>.

Di seguito viene riportato un esempio di utilizzo:

```
var distance = jaro_winkler.distance( 'text1', 'text2' );
```

7. NLP, Natural Language Processing

In questo capitolo esaminiamo il componente più importante del progetto, il modulo di NLP. Tale modulo si occupa di gestire l'output prodotto dal server, in seguito ad una richiesta di Speech-Tagging, e di processare le informazioni ottenute al fine di definire una struttura dati in grado di rappresentare in un modo semplice le informazioni utili relative al testo riconosciuto durante la fase di Speech Recognition.

Da quanto detto, si intuisce che il testo riconosciuto dal modulo di speech viene inviato al server attraverso una richiesta GET HTTP, richiedendo una elaborazione di Speech Tagging. Tale richiesta viene svolta utilizzando una chiamata Ajax, con una sintassi semplificata grazie all'utilizzo della libreria [jQuery](#):

```
var nlp = null;

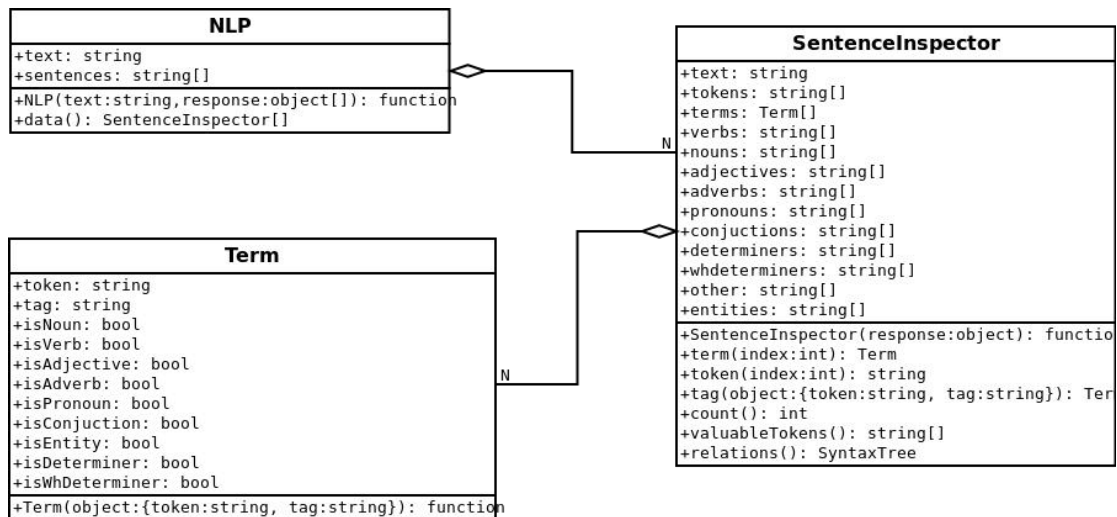
var ajaxRequest = $.ajax({
  'url': 'nlp',
  'data': {
    text: 'hello world'
  },
  'headers': {'Accept': 'application/json'},
  'success': function( response ){
    nlp = new speech2owl.NLP( 'hello world', response );
    // Do something
  },
  'error': function( response ){
    // Do something
  }
});
```

Come già specificato, il server risponderà con una serializzazione json:

```
[{
  "sentence": "hello world",
  "terms": [
    {"token": "hello", "tag": "UH"},
    {"token": "world", "tag": "NN"}
  ]
}]
```

Da notare che l'oggetto restituito è un array di oggetti, questo perchè il server si occupa di suddividere il testo, passatogli come argomento, in frasi costituenti e di eseguire lo Speech-Tagging singolarmente su ognuna di queste.

A questo punto esaminiamo da vicino la struttura dei vari componenti del modulo NLP realizzato.



Esaminando il diagramma UML si nota che l'oggetto di tipo `speech2owl.NLP` si occuperà di istanziare tanti oggetti di tipo `SentenceInspector` quante le frasi costituenti il testo da elaborare. Questi ultimi, infine, conterranno tanti termini quanti i token individuati nella frase dalla fase di Speech-Tagging, dove in base al tag avremo informazioni dettagliate sulla natura grammaticale di ognuno di questi.

8. Annotazione delle entità

L'operazione di annotazione del testo è stata implementata in un modulo del progetto, denominato `EntityLinker`.

Tale modulo si occupa di automatizzare l'operazione di estrazione di triple dall'ontologia e di trovare i match migliori.

Come per i moduli precedenti, abbiamo una fase di inclusione:

```
<script src="speech2owl/entitylinker.js"></script>
```

Mentre la fase di annotazione è richiamabile nel seguente modo:

```
var linker = new speech2owl.EntityLinker( nlp, rdf, true );
var links = linker.link( subject, predicate, object, confidence );
```

- ✓ `nlp` è un argomento del tipo `speech2owl.NLP`.
- ✓ `rdf` è un argomento di tipo `speech2owl.RDF`.
- ✓ Il terzo argomento è un booleano che serve ad abilitare o disabilitare i log del modulo

Il metodo `link` restituisce un array di triple, i match migliori rispetto all'ontologia di riferimento. Anche qui il parametro `confidence` è facoltativo, se non specificato, questo viene impostato a 0,8.

A questo punto partendo dal testo in chiaro e le entità individuate dall'`EntityLinker`, è possibile eseguire l'annotazione del testo che ci produrrà il

seguente risultato (la parte di codice che svolge questa operazione di rappresentazione grafica delle annotazioni è situata nel file `index.html`, da riga 175 a 237):

☐ Continuous speech

Confidence

Annotate Ontology Lite Ontology Full

<http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza>

hasBase some PizzaBase

John is a boy which likes eating pizza Margherita. Sarah likes pizza too, but she prefers Capricciosa

Da notare che all'interno dei popup rispettivi ad ogni entità individuata, vengono riportate le proprietà ad esse associate.