1. **INTRODUZIONE**

Il tema d’anno svolto consiste nell’apportare modifiche nel progetto open source WEBVOWL (MIT license).

Le modifiche vertono sull’aggiunta delle feature di modifica dell’ontologia visualizzata, rendendo possibile quindi l’inserimento, la cancellazione o la modifica di un elemento.

La struttura del progetto WEBOWL è, nel suo insieme, ovviamente orientata alla visualizzazione dei dati e non alla loro modifica e, pertanto, è stato necessario sviluppare moduli per la modifica ed integrarli in quello che è il progetto originale. È stata trattata una restrizione degli aspetti di un’ontologia che è possibile modificare.

1. **WEBVOWL**

WEBVOWL è una applicazione web atta alla sola visualizzazione di ontologie. Implementa la visualizzazione delle notazioni per le ontologie OWL attraverso un grafo orientato che rappresenta l’ontologia stessa.

E’ bene specificare che l’applicativo lavora convertendo i file owl in formato vowl (formato accettato). La conversione avviene automaticamente in caso di IRI o URI mentre la visualizzazione di una ontologia in locale è possibile convertendo dapprima il file attraverso il software owl2vowl, applicativo java, scaricabile dal sito web di webvowl.

Webvolw, ad oggi, non permette la visualizzazione di tutti i costrutti di OWL2 o la combinazione di essi.

**2.1 Struttura del progetto**

L’applicativo è del tipo client side (utilizzo del solo JAVASCRIPT) ed è modularizzato mediante node.js.

Nei sorgenti, oltre ai file di stile css, sono presenti diversi script javascript suddivisi nelle cartelle webvowl e app. Nella cartella app sono presenti gli script per la gestione della UI (sidebar, menu, etc) mentre nella cartella webvowl sono presenti gli script relativi all’interpretazione e alla visualizzazione dell’ontologia.

Importanti tra questi sono:

* app/js/app.js: si occupa di inizializzare la pagina e di caricare l’ontologia.
* webvowl/js/graph.js: si occupa della rappresentazione dell’ontologia attraverso un grafo.
* Webvowl/js/parser.js: si occupa del parsing dell’ontologia e della relativa rappresentazione interna.
* Gruntfile.js: si occupa di importare nella cartella deploy tutti i file esterni e di supporto necessari al funzionamento dell’applicativo.
* Index.html: corpo della pagina web di webvowl

Nella cartella deploy sono contenuti i sorgenti compilati.

1. **MODIFICHE APPORTATE**

Essendo il progetto originale orientato alla sola visualizzazione di ontologie si è reso necessario apportare modifiche ai sorgenti originali e sviluppare ulteriori script esterni che abilitassero anche la modifica.

Per assicurare una facilità di integrazione delle modifiche sviluppate in future versioni, si è cercato di limitare i cambiamenti ai file originali, preferendo lo sviluppo di moduli esterni.

In particolare, dei file originali sono stati modificati:

* Graph.js: si è inserito uno script che abilitasse la cattura del click del tasto destro sul nodo e visualizzazione del relativo menu contestuale (righe 810-857). È stato inoltre aggiunta la referenza al nodo cliccato dall’utente per la sua relativa elaborazione.
* Index.html: si è modificata la struttura dell’idex.html per la disabilitazione del menu contestuale di default del browser, integrato jQuery e la cattura della posizione del click del mouse (194-206).
* App.js: esternalizzazione della funzione “loadOntologyFromText” in modo tale sia accessibile dai moduli da noi creati (143-199).
* Gruntfile.js: sono state aggiunti tra gli “import” i file .css, .js e .html da noi creati (69,75,76).
* Parser.js: data la natura differente della elaborazione dei dati che è necessario effettuare nel caso della sola visualizzazione di una ontologia piuttosto che anche della sua modifica, sì è scelto di sviluppare un modulo esterno (myparser.js) che trattasse i dati in modo tale da rendere possibile anche la modifica degli stessi.

Sono stati sviluppati i seguenti moduli:

* Myparser.js: script che si occupa di effettuare un re-parsing dell’ontologia e implementa i metodi per modificarla ed il reperimento dei dati dalla stessa.
* Page.js: script atto alla creazione della pagina popup, mostrata all’utente, per la modifica (insert, edit, delete di un nodo).
* pop.html: pagina popup da mostrare all’utente.
* Css vari per lo styling della pagina.

**3.1 Graph.js**

Lo script utilizzato (presente in src/js), è un plugin per la creazione di menu contestuali ampiamente configurabile e per scopi multipli, chiamato contextMenu.js. Per il suo funzionamento si affida alla libreria jQuery che è stata importata anch’essa nel progetto e viene esclusivamente utilizzata per il corretto funzionamento dello script menu contestuale.

Per quanto riguarda l’integrazione di tale menù all’interno del progetto, all’interno del file graph.js, che si occupa della gestione del grafo visualizzato, abbiamo individuato la sezione di codice inerente i “click listenerer” dei nodi del grafo ed integrato il codice summenzionato.

In questo modo è stato possibile abilitare il click del tasto destro sui singoli nodi e far apparire il relativo menu contestuale. In particolare, la modifica consiste nell’inserimento di una funzione (myContextMenu()) nel cui interno vi è dichiarato un array di oggetti(menu), uno per ogni voce che si vuole visualizzare.

Ogni oggetto si compone dei campi name, title e fun. Il campo fun è stato modellato di modo tale che al click su di esso, venga visualizzata la finestra popup relativa alla modalità add, edit e delete di un nodo.

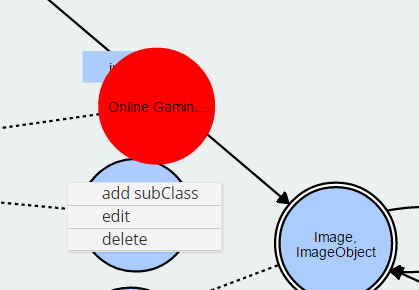


Figura - click destro sul nodo Online Gaming

**3.2 App.js**

Questo script, come detto precedentemente, si occupa dell’inizializzazione della pagina.

Cosa più importante è il compito di caricare il file JSON, parsarlo e validarlo, quindi passarlo alle strutture che si occuperanno della visualizzazione dello stesso.

Si è reso necessario creare una copia dell’istanza del modulo app e l’esternalizzazione della funzione loadOntologyFromText da richiamarsi all’interno dei nostri script. Questa necessità è dovuta al fatto che un import del modulo app all’interno degli script causava degli import ciclici che causavano degli errori a runtime. La funzione loadOntologyFromText è stata essenziale al fine di caricare, all’interno del sistema, l’ontologia a cui sono state apportate le modifiche effettuate dall’utente. Sfruttando questa funzione è stato possibile ridisegnare il grafo dell’ontologia ad ogni modifica effettuata, evitando che il file venisse salvato e ricaricato manualmente. Questa funzione è stata richiamata passando come parametro il json aggiornato in formato di testo, in modo tale che il sistema lo validasse e convertisse secondo la propria rappresentazione interna.

**3.3 Page.js**

Script da noi creato che si occupa della creazione della finestra popup in base al click sul menu contestuale.

Le voci del menu, come raffigurato in fig.1, sono 3:

* Add subClass;
* Edit;
* Delete.

In base alla voce selezionata la visualizzazione è differente. Selezionando add subClass si abilita la modalità insert di un nuovo nodo (di default figlio del nodo su cui si è cliccato). Viene visualizzato l’id del nuovo nodo che si sta creando mostrando all’utente il form da popolare con i dati relativi al nodo.

Selezionando, invece, la voce edit, innanzitutto vengono reperite tutte le informazioni sul nodo selezionato (myparser) e quindi visualizzate in un form identico a quello dell’inserimento, mentre selezionando delete, viene caricato una finestra popup che ci permette di eliminare il nodo selezionato.

Ebbene specificare che, data la natura di webvowl orientata alla sola visualizzazione, in caso di nodi equivalenti, questi ultimi vengono rappresentati come un unico nodo e viene meno la possibilità di cliccare sul singolo nodo equivalente.

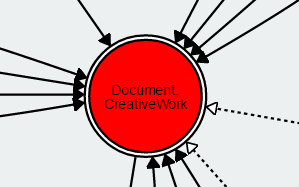
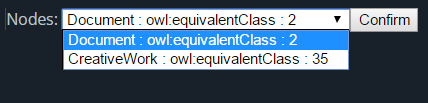


Figura - rappresentazione grafica di due nodi equivalenti

Come è possibile vedere in figura due, il nodo Document è equivalente al nodo CreativeWork. Cliccando su tale nodo vengono reperite le sole informazioni del nodo Document rendendo impossibile, di fatto, editare e cancellare direttamente CreativeWork.

Per poter operare su CreativeWork sarebbe stato necessario dapprima scindere la relazione di equivalenza (in modo tale che fossero rappresentati come singoli nodi) e quindi operare sul nodo di interesse.

Per ovviare a questa inconvenienza, all’interno dello script page, viene dapprima analizzato il nodo e, se in modalità di editing o cancellazione, si è in presenza di un nodo che abbia equivalenze verrà mostrata una select che permetta all’utente di specificare il nodo su cui operare(fig.3-4).



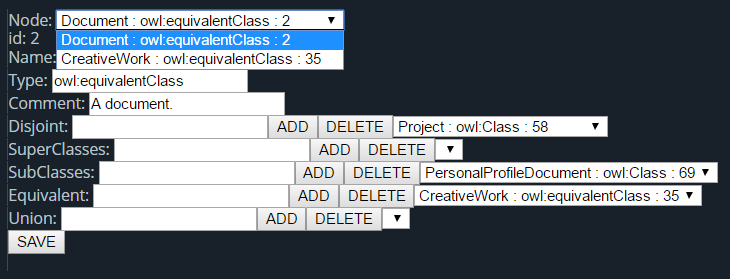


Figura - Selezione del nodo da eliminare e modificare

**3.3.1 Rappresentazione dei dati all’utente**

Come è possibile vedere in figura 3 sì è scelto di rappresentare all’utente i nodi dell’ontologia nel formato *Name : Type : ID* dove:

* Name: è il nome nel linguaggio corrente per il nodo selezionato;
* Type: è il tipo owl del nodo selezionato;
* Id: numero identificativo univoco del nodo selezionato all’interno del JSON.

La scelta di questa rappresentazione è stata fatta per distinguere nodi con lo stesso nome ma con iri differenti (avranno id differenti, vedi nodo Person in foaf). Come ulteriore supporto all’utente nell’identificazione del nodo corretto si è riutilizzata la features di highlight node già presente in graph.

In questo modo, durante la selezione di un nodo esso verrà cerchiato in rosso nel grafo.

4 My parser

4.1 strutture dati

4.2 funzioni varie

4.3 insert

4.4 edit

4.5 delete

4.6 read

3.3.2 Rsottoparagrafo di myparser

Diciamo che la rappresentazione interna di web owl era spuria ed inadatta alla modifica e parliamo delle nostre strutture e come si è operato