Relazione progetto di Editoria Elettronica (A.A. 2015/2016)

Traccia 2: SVG+Javascript

Michele Mallia (matr. 480650)

Presentazione:

Il progetto realizzato dal candidato Michele Mallia è un piccolo programma realizzato in linguaggio Python che prende in input un file .svg e restituisce un file out-put, in formato .html, che contiene il codice svg stesso del file in input più una parte in Javascript per le seguenti operazioni: il recupero dei valori (get) degli elementi che compongono il disegno vettoriale e il settaggio (set) dei parametri tramite un input stabilito dall'utente. Una volta che l'utente setta i valori degli attributi che vuole modificare, il disegno si modifica in tempo reale.

Funzionamento del programma:

1. Implementazione dell'interfaccia DOM:

Alla base del progetto c'è la manipolazione del DOM del file in input preso in considerazione. Per fare questo c'è bisogno della libreria **xml.dom.minidom**. Con questa libreria Python ci da la possibilità di visitare il DOM di un file preso in input e poter analizzare i suoi nodi per recuperare informazioni importanti. Rispetto alla libreria madre, **xml.dom**, contiene meno caratteristiche, ma contiene le funzionalità necessarie per gli scopi del progetto.

2. Analisi del file .svg:

Dopo aver importato la libreria, il passo successivo eseguito dal programma è quello di analizzare il codice xml presente nel file vettoriale in input. Viene applicata la funzione parse() al file in input che restituisce un oggetto Document, ovvero il DOM del file preso in analisi. Fatto questo, si deve dare al programma l'istruzione per ottenere la radice dell'albero con l'istruzione root = dom.documentElement.

3. Ricerca e raccolta dei nodi figli tramite il metodo ricorsivo:

Ottenuta la radice dell'albero è ora possibile accedere al valore dei nodi figli. Per visitare tutti i nodi figli dell'albero si usa una funzione ricorsiva nella seguente forma:

```
def child_attr(root, at=[]):
   if root.childNodes:
       at = at
       for node in root.childNodes:
          if node.nodeType == node.ELEMENT_NODE:
               at.append(node.attributes.items())
               child_attr(node)
   return at
```

Se il nodo in input ha dei figli, si esplora la lista dei figli fin quando non si trova un nodo che corrisponde a un nodo elemento dell'albero (node.nodeType restituisce uno short che va da 1 a 12, se restituisce 1 significa che corrisponde a un nodo elemento); se la condizione è vera, appende il nodo (o un suo valore) nella lista e richiama ricorsivamente la funzione. In questo modo vengono esplorati tutti i nodi dell'albero, a prescindere dal loro livello di profondità del nodo.

Ci sono cinque funzioni ricorsive nel programma con questa forma:

- **child_attr** serve a pescare tutti i valori relativi agli attributi di un nodo; la funzione inizializza un array vuoto e "appende" gli attributi del nodo con node.attributes.items(), che restituisce una tupla composta dal

nome dell'attributo con il suo valore (es: [('style','fill: black; stroke-width: 100...ecc...')]); alla fine la funzione restituisce una lista con tante tuple quanti sono i nodi dell'albero con attributi.

- **child_name** serve per recuperare tutti i nomi dei nodi dell'albero (node.TagName); anche questa funzione viene inizializzata con la *root* dell'albero e un array inizialmente vuoto.

- **getStoredId** serve per recuperare degli eventuali *id* presenti nel vettoriale; questa funzione è molto importante perché capita spesso che alcuni nodi facciano da riferimento ad altri, e sarebbe preferibile non dover settare nuovi id andando incontro al rischio far saltare alcuni collegamenti fra gli elementi del disegno (ad esempio capita di trovare degli elementi, come un cerchio o un rettangolo, che fanno riferimento a gradienti o filtri; far saltare questo riferimento comprometterebbe la visualizzazione del disegno vettoriale). La funzione restituirà un array con tutti gli id presenti nel vettoriale.

- **setId** serve per dare un id a tutti i nodi dell'albero, anche a quelli che ce l'hanno già; questa funzione ricorsiva settare un id a un nodo ci permettere di poter raggiungere un solo unico elemento dell'albero e di poterlo modificare o visitare. Non restituisce alcun array.

- **getId** serve a recuperare tutti gli id presenti nell'albero, quindi recupera anche i riferimenti da inserire nella parte di codice Javascript per poi poter fare le operazioni di recupero e settaggio dei valori dei nodi. Restituisce un array di stringhe.

4. Pulizia dei dati acquisiti

I dati dell'albero che sono stati acquisiti si trovano in quattro array (**tagName**, **attri**, **storedId** e **get_Id**), ognuno specializzato per contenere un determinato tipo di valore. Una volta ottenuti i dati, è necessario applicare delle regole di pulizia secondo determinati criteri:

1. Vanno eliminati i dati relativi a nodi che non hanno attributi da modificare;
2. Vanno recuperati gli id dei nodi che erano già presenti prima del settaggio tramite **setId**;
3. Per ogni modifica, vengono popolate nuove liste (**newTag, newAttr, newId, newStoredId**) che contengono i dati dei nodi che hanno superato le condizioni di filtraggio.

Con questi dati più "raffinati" possiamo compiere delle operazioni utili, come il reimpostare i vecchi id dei nodi (e salvaguardare eventuali riferimenti presenti fra i nodi; operazione fatta mediante Javascript) e reinserirli nella nuova lista degli id per poter poi richiamare ogni nodo correttamente.

5. Creazione del file HTML

Per poter far convivere svg e javascript assieme si è deciso di creare un file html che contenesse entrambi i codici, incorporati alla pagina stessa.
Per prima cosa il programma legge il file in input e imposta le istruzioni per la scrittura del file in output.

Vengono poi stampate le intestazioni per il documento html tramite **write()** di Python, così come anche la parte di Javascript. Per copiare il codice xml nella pagina web è bastato inserire **str(dom.toxml())** che restituisce una stringa contenente il DOM del file convertito in xml (non indentato, per quello si può usare la funzione **toprettyxml()** che restituisce il codice indentato).

La struttura del file in uscita è abbastanza basilare: c'è la dichiarazione della DTD, il body, e due div per contenere, in uno, i nomi dei nodi, e nell'altro i nomi degli attributi con i rispettivi valori.

6. Javascript e la parte dinamica

Per incorporare il linguaggio Javascript è bastato trascrivere la stringa del codice da inserire all'interno di **write()**.

Con Javascript si modifica il DOM della pagina html e si possono creare dinamicamente elementi come bottoni, paragrafi con nome e/o valore degli attributi e input di tipo text tramite i dati dei nodi acquisiti con Python.

Viene fatto un ciclo all'interno delle liste Python e vengono scritte tante righe di codice js quante sono le informazioni acquisite dai nodi del disegno vettoriale.

La pagina html si presenta con il disegno vettoriale (preso in input) nella parte superiore della pagina e, al centro, con i nomi degli nodi elemento posti in colonna, ognuno con due pulsanti **'get'** e **'set'** per settare o ricevere il valore degli attributi di un singolo elemento.

Entrambe le funzioni sono simili e sono definite da **getAttr()**, **setAttr()**: viene scritta una funzione con un nome identificativo diverso accompagnato dall'id dell'elemento in questione (es: 'getAttrrect0', 'getAttrgrad1' ecc...) in modo da non dover creare funzioni con lo stesso nome e quindi creare confusione.

Quando un utente preme sul pulsante set viene riportato un elenco di attributi modificabili singolarmente accompagnati ciascuno da un bottone che richiama una funzione **setVal()** al suo click; viene preso il valore dell'utente preso dalla input text e viene inserito nel nodo elemento tramite il suo richiamo tramite **getElementById()**.

Conclusioni:

Con questo progetto è possibile visualizzare i disegni vettoriali, i suoi dati e modificare i suoi valori visualizzando in tempo reale la modifica del disegno.

Questo progetto può servire a capire come è fatto un disegno vettoriale e di come si comportano i suoi componenti in base alle modifiche dell'utente, rendendo facile, con un'impostazione quasi da gioco, la comprensione di una tecnologia ampiamente usata sia nel campo della grafica, sia nella programmazione web.