Università degli Studi di Firenze
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Anno Accademico 2011/2012
Discussione Tesi di Laurea

Visualizzazione Grafica di Algoritmi in HTML5

Laureando:
Tommaso Papini
tommaso.papini.unifi@gmail.com

Relatore:
Pierluigi Crescenzi
pierluigi.crescenzi@unifi.it

[Frame 1] Buongiorno a tutti, sono Tommaso Papini, e la mia Tesi si intitola *Visualizzazione Grafica di Algoritmi in HTML5.*

In questa tesi è stato analizzato il problema della visualizzazione degli algoritmi ed è stato sviluppato il quarto aggiornamento per il sistema di visualizzazione di algoritmi *AlViE*, con l'obiettivo principale di poter esportare le visualizzazioni prodotte in pagine Web, utilizzando gli strumenti messi a disposizione dal nuovo standard HTML5.

[Frame 2] La visualizzazione di algoritmi è una branca dell'Informatica che si occupa del problema della rappresentazione grafica dell'esecuzione di programmi ed algoritmi, al fine di semplificarne la comprensione. Quindi i principali obiettivi della visualizzazione di algoritmi sono principalmente pedagogici, ma possono anche essere di natura più tecnica, come ad esempio possono fornire un utile strumento per il debugging.
[Frame 3] I primi sistemi di visualizzazione di algoritmi, BALSA e TANGO, nacquero negli anni '80 ed influenzarono tutti i sistemi successivi, che implementarono sempre più funzioni rispetto ai loro antenati, come il supporto alla grafica a colori o 3D, la visualizzazione di programmi concorrenti o l'esecuzione del sistema attraverso il Web, sino ad arrivare ai tre sistemi più recenti, JHAVÉ, Trakla ed OpenDSA, che vengono tutt'ora utilizzati ed aggiornati.

[Frame 4] È stato scelto il nuovo standard HTML5 per l'esportazione delle visualizzazioni sul Web in quanto esso offre nuovi strumenti per la gestione di applicazioni multimediali, come il controllo di flussi audio o video o la gestione di grafica 2D e 3D.
In particolare, in questo progetto, viene utilizzato lo strumento Canvas (che in inglese significa *tela*), messo a disposizione da HTML5, per la creazione e gestione di grafica bitmap 2D in tempo reale. È stato scelto Canvas, anziché uno strumento più complesso per la grafica, in quanto i disegni che compongono le visualizzazioni sono spesso figure geometriche elementari e Canvas

permette di disegnare queste figure tramite semplici primitive grafiche JavaScript e ad un bassissimo costo computazionale.

Come accennato prima, il sistema di visualizzazione di algoritmi utilizzato non è stato scritto da zero, ma ci siamo basati sul sistema AlViE, sviluppato ed aggiornato in questi ultimi anni dal Professor Crescenzi.
[Frame 5] AlViE, che sta per *Algorithm Visualization Environment*, permette la visualizzazione 2D a colori di algoritmi scritti in Java. Inoltre, AlViE presenta un discreto grado di interazione con l'utente, mettendo a disposizione pulsanti per la navigazione attraverso i vari passi (o step) di visualizzazione o la selezione di diversi tipi di zoom.
Il sistema AlViE memorizza su disco le visualizzazioni prodotte in specifici file XML, dove vengono indicate le strutture dati presenti ad ogni passo della visualizzazione e le caratteristiche grafiche di tali strutture.
[Frame 6] A titolo d'esempio, questa è una tipica schermata di AlViE, dove possiamo vedere, in alto, i bottoni di navigazione e nella parte centrale un passo dell'esecuzione dell'algoritmo del SelectionSort, con tanto di pseudocodice.

[Frame 7] Le principali novità introdotte in questa quarta versione di AlViE sono, come già accennato, la possibilità di esportare le visualizzazioni prodotte in pagine HTML, e l'introduzione di un editor di strutture dati che, con pochi click, permette all'utente di creare file di input tramite una semplice interfaccia tabulare.

[Frame 8] Per esportare le visualizzazioni prodotte su AlViE in pagine Web si è quindi sviluppato un Compilatore XML-HTML5 che traduca la specifica XML della visualizzazione nella corrispettiva specifica HTML, implementando opportunamente le funzioni Canvas messe a disposizione da HTML5.
Come possiamo vedere da questo schema, in linea di massima il compilatore è suddiviso in una fase di parsing ed una fase di compilazione.

[Frame 9] La fase di parsing rappresenta la fase in cui viene letto il file XML della visualizzazione e vengono estratte le informazioni in esso contenute; informazioni che verranno successivamente utilizzate per determinare la specifica HTML della visualizzazione.
Per quanto riguarda la scelta del parser, sono state analizzate due possibili soluzioni: utilizzando un parser XML generico, in questo caso il Digester, sviluppato dalla Apache, oppure utilizzando un parser ad-hoc, ovvero un parser scritto su misura per l'esportazione di visualizzazioni in pagine Web. Sono stati implementati in AlViE entrambi i parser e sono stati comparati i tempi di compilazione su svariati algoritmi: come si può vedere dalla tabella nella slide, utilizzando il Digester i tempi di compilazione aumentano sensibilmente rispetto al parser ad-hoc, quindi in AlViE4 è stato mantenuto

quest'ultimo.

[Frame 10] La fase di compilazione rappresenta la fase in cui vengono determinati i comandi grafici Canvas da scrivere nel codice HTML per costruire la visualizzazione.
Una fase intermedia, prima della compilazione vera e propria, è la fase di calcolo dei dati impliciti. Per dati impliciti si intendono tutti quei dati, di natura grafica, che sono necessari per formare la visualizzazione sulla pagina HTML ma che vengono omessi dalla specifica XML, in quanto ridondanti. Un esempio si ha quando deve essere disegnato un array: nella specifica XML viene indicato soltanto il punto di origine in cui disegnare l'array e la larghezza degli elementi, mentre per costruire la visualizzazione con Canvas è necessario disporre dei punti di origine di ogni singolo elemento.
A seconda di quando si decide di calcolare l'insieme di questi dati impliciti si parla di calcolo **a monte** o **a valle**, rispettivamente prima e dopo la compilazione vera e propria.
Nel caso del calcolo a valle si avrà che nel file HTML verrà scritta l'invocazione a una funzione JavaScript, opportunamente definita nell'intestazione della pagina, che calcolerà i dati impliciti ogni volta che il browser carica la pagine. nel calcolo a monte si ha invece che questi dati vengono calcolati dal compilatore, che potrà quindi scrivere sul file HTML direttamente le primitive grafiche corrispondenti.
La scelta di queste due soluzioni va a influire su tre fattori: il tempo di compilazione, le dimensioni della pagina Web e i tempi di caricamento da parte del browser. Scegliendo il calcolo a monte si avrà una compilazione più lenta, un caricamento della pagina più veloce ed una pagina, generalmente, più pesante, in quanto ogni invocazione a funzione viene sostituita con più primitive grafiche.
Per AlViE4 è stato scelto il calcolo a monte per una questione puramente teorica: è stato ipotizzato che siano molte di più le volte che una visualizzazione, o un passo di visualizzazione, venga caricato dal browser, rispetto alle volte che essa può essere esportata in pagina Web tramite il compilatore o scaricata da Internet. Difatti, dal punto di vista di prestazioni, le due soluzioni sono indistinguibili, in quanto si hanno differenze di pochi nanosecondi per i tempi di compilazione e caricamento e pochi KB per le dimensioni della pagina.

[Demo]Vorrei mostrarvi, adesso, un semplice esempio dell'utilizzo di AlViE4.....

[Frame 11] Volendo entrare più nel dettaglio, sono state implementate, all'interno del compilatore, delle specifiche classi per ogni tag, dette TagHandler, ognuna delle quali si occupa di gestire il tag corrispondente. Per ogni tag letto dal file XML, viene selezionato il TagHandler giusto tramite il design

pattern del factory. Quindi il gestore effettuerà la fase di parsing del tag, estraendone i dati, e invocherà, ricorsivamente, il TagHandler di tutti i tag interni presenti. Una volta raccolti i dati da tutti i tag interni, si potrà quindi passare alla fase di calcolo dei dati impliciti e alla conseguente compilazione e scrittura su file HTML.