

UNIVERSITÀ DEGLI STUDI DI FIRENZE
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica



Tesi di Laurea

VISUALIZZAZIONE GRAFICA DI ALGORITMI IN
HTML₅

TOMMASO PAPINI

Relatore: *Pierluigi Crescenzi*

Anno Accademico 2011-2012

RINGRAZIAMENTI

La lista per esteso delle persone che dovrei ringraziare alla fine di questo percorso sarebbe veramente lunga: molti mi hanno aiutato e supportato, anche con il più piccolo dei gesti.

Ringrazio la mia famiglia ed i miei amici, per aver creduto in me ed avermi spronato ad andare sempre avanti. Ringrazio la mia ragazza, Sara, per avermi sopportato ogni volta che ero in crisi riguardo all'Università.

Infine, vorrei fare un ringraziamento speciale a mia sorella, Melissa, per l'aiuto e i consigli che mi ha dato, e per tutto il tempo che mi ha dedicato.

Grazie.

INDICE

Introduzione	1
1 Visualizzazione di algoritmi	3
1.1 Sorting Out Sorting	3
1.2 La famiglia Balsa	4
1.2.1 Balsa	4
1.2.2 Balsa II	6
1.2.3 Zeus	7
1.3 La famiglia TANGO	8
1.3.1 TANGO	8
1.3.2 POLKA	9
1.3.3 SAMBA	10
1.4 Altri sistemi di visualizzazione di algoritmi	10
1.4.1 ANIM	11
1.4.2 Mocha	11
1.4.3 Leonardo	12
1.4.4 GAWAIN	13
1.4.5 Pavane	14
1.4.6 CATAI	15
1.5 Sistemi di visualizzazione di algoritmi recenti	16
1.5.1 JHAVÉ	16
1.5.2 Trakla	17
1.5.3 OpenDSA	18
1.6 Tassonomia dei sistemi di visualizzazione di algoritmi	19
1.6.1 Portata	21
1.6.2 Contenuto	22
1.6.3 Forma	25
1.6.4 Metodo	27
1.6.5 Interazione	30
1.7 Conclusioni	31
2 HTML5	33
2.1 La nascita di HTML5	33
2.1.1 XHTML2	33
2.2 Da HTML4 ad HTML5	34
2.2.1 Canvas	37
3 ALViE	43
3.1 Caratteristiche principali	43
3.1.1 Specifica delle visualizzazioni	45
3.1.2 Specifica delle strutture dati	45

3.2	AlViE4	46
3.2.1	HTML5	47
3.2.2	Editor di strutture dati	48
3.3	Un esempio pratico	49
3.4	Sviluppi futuri	51
4	Da AlViE ad HTML5	53
4.1	Specifica HTML5 delle visualizzazioni	53
4.2	Compilatore XML-HTML5	54
4.2.1	Fase di parsing	54
4.2.2	Fase di compilazione	56
4.2.3	Pesantezza della pagina vs. pesantezza di compilazione	57
	Conclusioni	59
	Bibliografia	61

ELENCO DELLE FIGURE

Figura 1	Linguaggio L6	4
Figura 2	Sorting Out Sorting: alcuni esempi	5
Figura 3	BALSA: alberi binari e bilanciati	6
Figura 4	La <i>electronic classroom</i> della Brown University.	6
Figura 5	Zeus: dimostrazione del Teorema di Pitagora	8
Figura 6	XTANGO: BubbleSort	9
Figura 7	POLKA: thread concorrenti in esecuzione	10
Figura 8	ANIM: InsertionSort e QuickSort a confronto	12
Figura 9	Leonardo: algoritmo di Kamada-Kawai	13
Figura 10	GAWAIN: MergeSort	14
Figura 11	Pavane: il QuickSort in 3D	15
Figura 12	JHAVÉ: QuickSort	17
Figura 13	Trakla: un esercizio sul QuickSort	18
Figura 14	OpenDSA: InsertionSort	19
Figura 15	Il primo livello gerarchico della tassonomia	20
Figura 16	Tassonomia della Portata	21
Figura 17	Tassonomia del Contenuto	23
Figura 18	Tassonomia della Forma	25
Figura 19	Tassonomia del Metodo	28
Figura 20	Tassonomia dell'Interazione	30
Figura 21	Canvas: un quadrato rosso	39
Figura 22	Canvas: una casetta rossa	40
Figura 23	Canvas: casetta rossa con testo	40
Figura 24	Indici di un albero binario nella specifica XML di AlViE	46
Figura 25	AlViE4: InsertionSort sul Web	47
Figura 26	Creazione manuale di un grafo orientato	49
Figura 27	AlViE4: visualizzazione del SelectionSort	51

INTRODUZIONE

Gli ultimi 20 anni sono stati caratterizzati da un utilizzo sempre più frequente delle tecniche di visualizzazione di algoritmi, da parte dei docenti e di Informatica, dal momento che esse costituiscono un importante strumento per l'apprendimento e la comprensione dei più svariati tipi di algoritmo.

Un altro strumento altrettanto importante, per i fini sopra indicati, è rappresentato dal Web. Sono infatti moltissime le applicazioni che permettono, a studenti e ricercatori di tutto il mondo, di scambiarsi e condividere dati e informazioni tramite Internet, facilitandone così l'apprendimento e la comprensione, affidandosi anche a strumenti multimediali come la produzione di un'immagine o di un video.

Il progetto *AlViE4* (*Algorithm Visualization Environment*, ovvero *Ambiente per la Visualizzazione di Algoritmi*) nasce appunto con l'intento di far incontrare queste due realtà, estendendo il programma *AlViE*, per la visualizzazione di algoritmi, al mondo del Web, tramite le nuove possibilità offerte dallo standard *HTML5*.

Questo lavoro si compone di quattro capitoli. Nel Capitolo 1, per cominciare, effettueremo una panoramica sulle tecniche di visualizzazione esistenti, soffermandosi in particolare sulla loro storia ed evoluzione. Inoltre, verrà proposta una tassonomia per la classificazione ed una maggior comprensione dei sistemi di visualizzazione di algoritmi.

Parleremo poi, nel secondo Capitolo, dello standard *HTML5* (*HyperText Markup Language*, ovvero *Linguaggio di Marcatura degli Ipertesti*): le novità introdotte rispetto alla versione precedente, le caratteristiche principali e le motivazioni che ci hanno spinto a scegliere gli strumenti messi a disposizione da questo nuovo standard per l'estensione di *AlViE* al mondo del Web.

All'interno del Capitolo 3 analizzeremo nello specifico il funzionamento del sistema *AlViE*, indicandone le novità introdotte nella quarta versione. Verrà fornito, in conclusione del Capitolo, un esempio passo-passo su come costruire un algoritmo interpretabile da *AlViE*, producendone quindi una visualizzazione. In conclusione, vedremo come è stato svolto il lavoro di estensione di *AlViE* al Web, indicando le tecniche di programmazione utilizzate ed i problemi riscontrati in fase di sviluppo.

VISUALIZZAZIONE DI ALGORITMI

“La visualizzazione (o animazione) di algoritmi è il processo di astrazione di dati, operazioni e semantica e della creazione di visualizzazioni grafiche dinamiche di queste astrazioni” [12].

Quindi la visualizzazione di algoritmi consiste nel rendere l'esecuzione di un dato algoritmo il più intuitiva possibile, da un punto di vista grafico, mostrando, tramite l'utilizzo di figure o etichette minimali, l'evoluzione dell'esecuzione dell'algoritmo e dei dati da esso gestiti.

Le tecniche di visualizzazione di algoritmi sono nate e si sono sviluppate al fine di agevolare l'insegnamento e la comprensione della maggior parte degli algoritmi. Infatti, la rappresentazione grafica e dinamica delle strutture dati gestite da un algoritmo e la loro evoluzione durante l'esecuzione di quest'ultimo, forniscono un importante strumento educativo in fase di comprensione e analisi di un algoritmo.

Inoltre, la visualizzazione di un algoritmo, risulta molto utile anche in fase di test e di debug, in quanto un possibile errore, se presente, risulterà immediatamente visibile al programmatore, che non dovrà quindi ricercare il problema analizzando i dati risultanti in formato alfanumerico o, in caso peggiore, il codice stesso.

Uno dei primissimi esempi di visualizzazione di algoritmi risale al 1966 quando Ken Knowlton produsse, per conto della Bell Telephone Laboratories, un video animato di circa 15 minuti in cui veniva mostrato il funzionamento del linguaggio *L6* [8]. Il linguaggio *L6* era un linguaggio di basso livello orientato a liste concatenate creato dai ricercatori dei Laboratori della Bell Telephone Company. Infatti la denominazione *L6* del linguaggio è in realtà un'abbreviazione che sta per *(Bell Telephone) Laboratories' Low-Level Linked List Language*, ovvero *Linguaggio di basso livello orientato a liste concatenate dei Laboratori Bell*.

1.1 SORTING OUT SORTING

Anche se, come abbiamo visto, i primi video di visualizzazione di algoritmi risalgono addirittura agli anni '60, quello che convenzionalmente viene considerato il primo ad aver introdotto la nozione di visualizzazione di algoritmi è un video del 1981, creato da Ronald Baecker presso l'Università di Toronto, che prende il nome di *Sorting Out Sorting*, inteso come *Capire gli Algoritmi di Ordinamento*

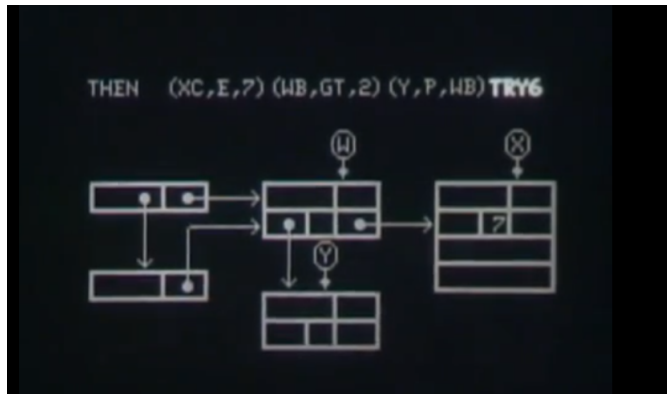


Figura 1: Uno snapshot preso dal video esemplificativo del linguaggio L6 di Ken Knowlton.

[20]. Questo filmato, infatti, ispirò moltissimi insegnanti ad adottare tecniche di visualizzazione di algoritmi per l'insegnamento di quest'ultimi e permise, di conseguenza, lo sviluppo e la ricerca per queste tecniche.

Questo video di 30 minuti mostrava, graficamente, il funzionamento di nove diversi algoritmi di ordinamento, operanti su diversi insiemi di dati, e ne comparava l'esecuzione.

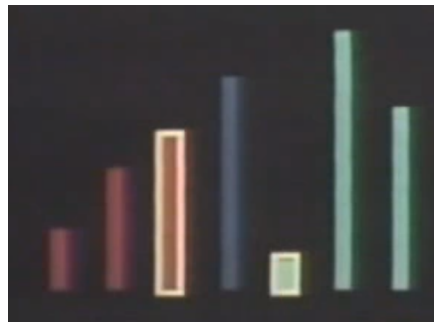
Dopo la pubblicazione del video *Sorting Out Sorting*, ed in particolare tra gli anni '80 e i primi del '90, si svilupparono due importanti sistemi nel campo della visualizzazione di algoritmi che influenzarono largamente tutti i sistemi successivi: i sistemi *BALSA* e *TANGO*.

1.2 LA FAMIGLIA BALSA

1.2.1 BALSA

BALSA (Brown Algorithm Simulator and Animator, ovvero Simulatore e Animatore di Algoritmi di Brown) è stato il primo sistema di visualizzazione di algoritmi largamente conosciuto. *BALSA* fu sviluppato da Marc Brown e da Robert Sedgewick presso la Brown University e permetteva la visualizzazione interattiva di programmi, con tanto di visualizzazione multipla di algoritmi e strutture dati [16]. Inizialmente venne utilizzato nei laboratori informatici della Brown University (Brown's electronic classroom) come strumento interattivo per l'insegnamento degli algoritmi.

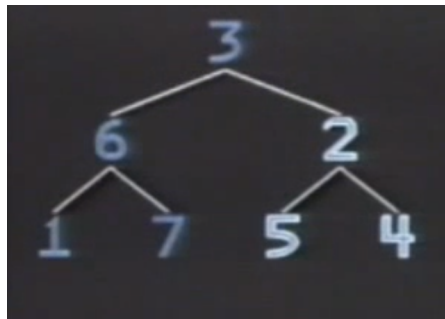
Il concetto chiave attorno al quale si sviluppa il sistema *BALSA* era quello



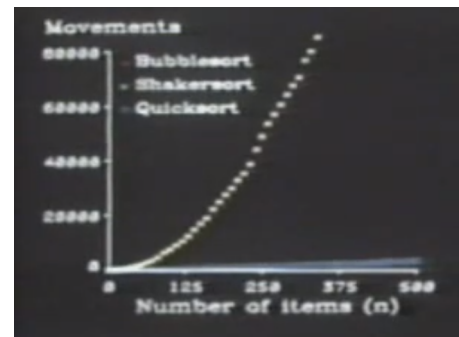
(a) InsertionSort



(b) BubbleSort



(c) HeapSort



(d) Una comparazione di tre algoritmi di ordinamento

Figura 2: Alcune immagini tratte dal video Sorting Out Sorting

degli *eventi interessanti* (dall'inglese, *interesting events*): l'esecuzione dell'algoritmo in ambiente BALSA produceva una serie di eventi, che venivano poi visualizzati in una finestra in formato grafico. Per eventi interessanti si intendono quindi operazioni, effettuate dall'algoritmo stesso, come la modifica del valore di una variabile, uno scambio di valori, l'allocazione o la liberazione di memoria, ecc.

Uno dei problemi principali del sistema BALSA era l'estrema complessità in fase di preparazione di un'animazione: erano richieste in media dalle 15 alle 25 ore per scrivere il codice relativo all'algoritmo e alle visualizzazioni di ogni suo passo [1].

Un altro limite ascrivibile a BALSA, era poi quello di trovare una concettualizzazione, o modello, appropriata per le strutture dati e l'algoritmo, come anche di determinare una giusta "scala" per rappresentare i dati della visualizzazione corrente. Un albero, ad esempio, poteva crescere in maniera esagerata in alcuni algoritmi: ad ogni passo esso veniva ridisegnato dal sistema BALSA e questo

si traduceva, la maggior parte delle volte, in un aumento della complessità dell'intero sistema, nonché in una lettura più difficoltosa da parte dell'utente.

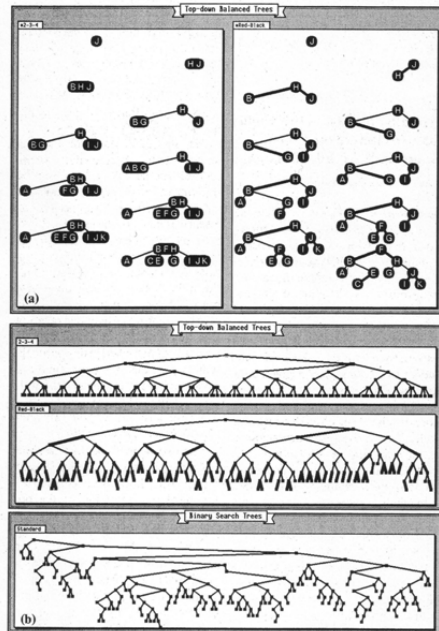


Figura 3: L'immagine di un'animazione in Balsa mentre vengono mostrati vari esempi di alberi binari e bilanciati.

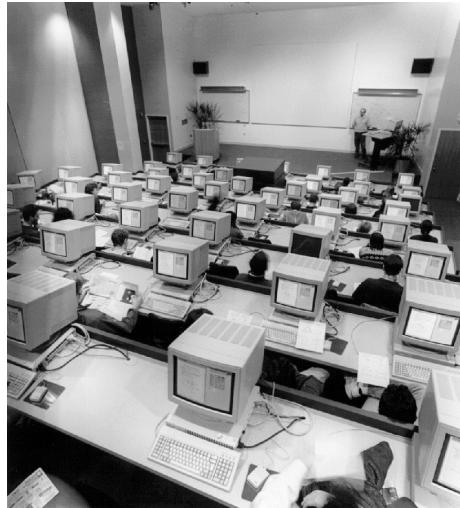


Figura 4: La *electronic classroom* della Brown University.

1.2.2 Balsa II

Balsa II fu il successore del sistema Balsa, anch'esso sviluppato da Marc Brown [17], che, come il predecessore, era orientato al concetto di "eventi interessanti" e di visualizzazioni multiple di algoritmi e/o strutture dati.

Il modello di animazione in Balsa II consisteva in un algoritmo, un generatore di input ed una o più visualizzazioni.

L'utente aveva la possibilità di visualizzare simultaneamente più algoritmi operanti sullo stesso input set, per poterne effettuare una comparazione, oppure cambiare, con pochi click, l'input dell'algoritmo, per effettuare test successivi. Inoltre questo sistema permetteva una, seppur minima, interazione con l'utente, che poteva, ad esempio, cambiare il valore di una variabile o la posizione di un oggetto grafico durante la simulazione dell'algoritmo [1].

Il limite principale che accomunava Balsa II al suo predecessore era la scarsa elasticità e maneggevolezza del linguaggio di programmazione supportato,

Pascal. L'utente infatti si ritrovava spesso in difficoltà a dover implementare il proprio codice in ambiente Balsa II.

Inoltre, il sistema Balsa II mancava sia di un editor grafico che di appropriate librerie grafiche, il che obbligava il programmatore a dover definire gli oggetti e le animazioni utilizzando primitive grafiche che risultavano molto scomode, specialmente per visualizzazioni complesse.

1.2.3 Zeus

Il sistema Zeus, sviluppato da Brown nel 1991 presso il Digital Equipment Research Center [18], segnò la fine del sistema Balsa II.

Zeus fu ideato prendendo in considerazione gli emergenti linguaggi di programmazione orientati agli oggetti. Tuttavia il linguaggio scelto da Brown per il sistema Zeus, Modula 3, non riscosse un gran successo, facendo rimanere il progetto Zeus un semplice prototipo di ricerca, senza mai trovare reale applicazione in ambito educativo. Inoltre Zeus fu implementato in un ambiente multi-processore e multi-thread, permettendo quindi l'esecuzione di programmi paralleli [1].

Come nei sistemi precedenti appartenenti alla famiglia Balsa, anche Zeus era orientato agli eventi: l'esecuzione dell'algoritmo in ambiente Zeus produceva una lista ordinata di eventi, ognuno dei quali implicava un aggiornamento della visualizzazione da parte del gestore grafico. In questo caso vennero inclusi anche eventi di tipo sonoro, che permisero di accentuare ulteriormente alcuni cambiamenti o animazioni particolarmente interessanti.

Questo sistema permise, rispetto ai suoi predecessori, non soltanto visualizzazioni a colori, ma anche visualizzazioni 3D.

Infatti la programmazione della visualizzazione avveniva utilizzando funzioni ad alto livello definite da librerie grafiche 2D o 3D utilizzando un linguaggio chiamato Obliq. La grafica veniva eseguita e gestita su un sistema X-Windows.

Un'altra importante novità di Zeus è l'introduzione dello pseudocodice all'interno delle visualizzazioni. Lo pseudocodice è il codice dell'algoritmo da eseguire, scritto in un linguaggio di programmazione fittizio, riportandone soltanto le operazioni principali o comunque di interesse. Lo pseudocodice è quindi un oggetto grafico di tipo testuale le cui righe vengono evidenziate durante l'esecuzione dell'algoritmo a seconda di quale operazione viene eseguita in quel preciso step.

Questo forniva all'utente una visione immediata della corrispondenza tra visualizzazione ed esecuzione dell'algoritmo, aiutando ulteriormente la comprensione di quest'ultimo.

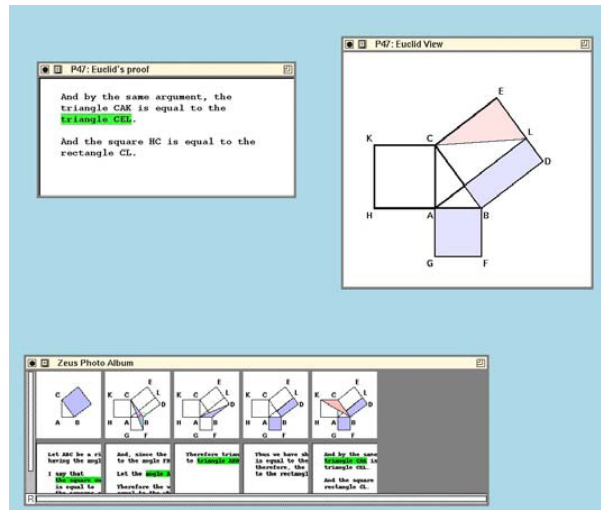


Figura 5: Un'immagine del sistema Zeus durante la visualizzazione della dimostrazione del Teorema di Pitagora.

1.3 LA FAMIGLIA TANGO

1.3.1 TANGO

L'acronimo TANGO sta per *Transition-based Animation GeneratiOn*, ovvero *Generazione di Animazioni orientata alle Transizioni*, e fu sviluppato da John Thomas Stasko verso la fine degli anni '80.

Il sistema TANGO presentava degli spiccati miglioramenti in quanto a grafica ed animazioni: le figure erano colorate e potevano essere evidenziate a seconda della loro importanza; le animazioni invece furono rese più fluide e standardizzate, utilizzando per esse un modello formale che prese il nome di *Path-Transition Paradigm*.

Inoltre venne ampiamente semplificata, rispetto al sistema BALSÀ, la fase di sviluppo di un algoritmo e della relativa visualizzazione.

Fu inoltre sviluppata una versione X-Windows di TANGO, denominata XTANGO [13], che permette al programmatore di utilizzare funzioni di alto livello, definite dal sistema X-Windows, per la parte grafica della visualizzazione, evitandogli quindi di dover programmare utilizzando primitive grafiche di basso livello.

Entrambe le versioni, TANGO e XTANGO, implementavano due processi separati comunicanti: uno per l'esecuzione dell'algoritmo ed uno per la gestione delle animazioni. Questo impediva la visualizzazione di programmi paralleli, e per farlo fu necessario modificare la struttura del sistema TANGO.

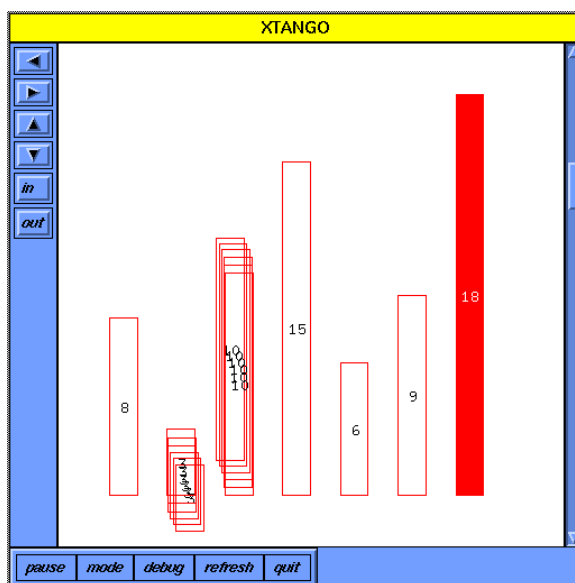


Figura 6: Un passo dell'esecuzione dell'algoritmo del BubbleSort in ambiente XTANGO che mostra l'utilizzo di più frames nella visualizzazione.

1.3.2 POLKA

In base alle considerazioni fatte nella Sezione precedente, per poter visualizzare programmi paralleli si rese necessario modificare la struttura dei sistemi di visualizzazione TANGO e XTANGO: fu così che venne sviluppato POLKA[1]. Il sistema POLKA nacque quindi per superare i limiti insiti nei sistemi di visualizzazione TANGO e XTANGO, e per poter rendere possibile la visualizzazione di programmi paralleli.

Una delle principali differenze con POLKA e XTANGO è che POLKA risulta essere più elegante, in quanto il sistema utilizzava il linguaggio C++, orientato agli oggetti, mentre XTANGO utilizzava il C.

Inoltre viene modificato il paradigma utilizzato per le animazioni: mentre in XTANGO il movimento di un oggetto grafico viene definito come un'azione atomica, nel sistema POLKA era presente una sorta di clock globale interno che permetteva di visualizzare il frame corrispondente ad ogni visualizzazione in quell'istante.

Come in XTANGO, i comandi grafici vengono generati per un sistema X-Windows. Vengono inoltre forniti dei comandi specifici per poter mettere in pausa la visualizzazione corrente, riavvolgerla o cambiare la velocità con cui vengono mostrati i vari frame.

Nella versione di base di POLKA le animazioni erano unicamente in 2D, ma

venne successivamente distribuita una versione, denominata POLKA 3D, che mise a disposizione visualizzazioni 3D e primitive 3D come sfere, coni, cubi e altro [6].

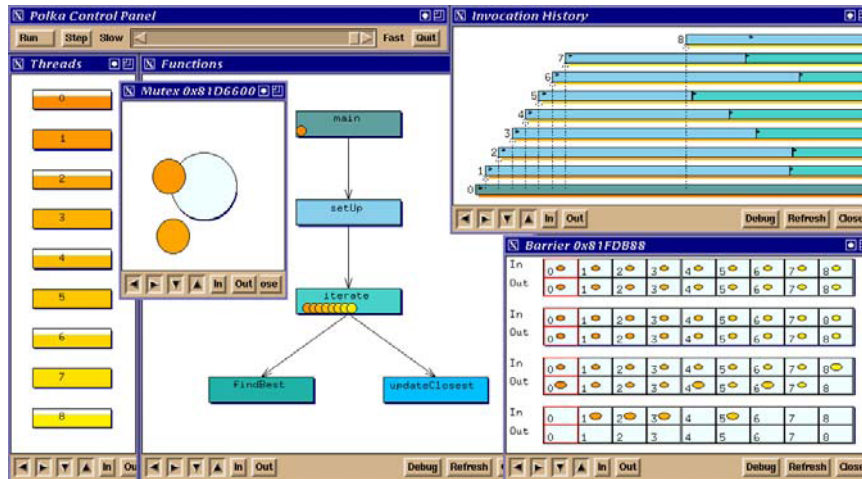


Figura 7: Visualizzazione di thread concorrenti in esecuzione in ambiente POLKA.

1.3.3 SAMBA

Venne sviluppato, per essere utilizzato in coppia con il sistema POLKA, un'interfaccia di front-end, denominata SAMBA, per la programmazione della parte grafica della visualizzazione.

Samba metteva a disposizione un insieme di comandi molto semplici che l'utente poteva utilizzare per costruire l'animazione. L'utente doveva generare un file ASCII contenente i comandi relativi all'animazione, il quale veniva passato all'interprete grafico di POLKA, che si preoccupava di generare l'animazione.

L'approccio introdotto con SAMBA, ovvero quello di separare il motore grafico dal codice dell'algoritmo, viene detto *animation scripting*.

1.4 ALTRI SISTEMI DI VISUALIZZAZIONE DI ALGORITMI

Vediamo in questa Sezione altri sistemi per la visualizzazione di algoritmi che, pur non appartenendo alle famiglie dei sistemi Balsa o TANGO, presentano innovazioni interessanti rispetto ai sistemi già analizzati.

1.4.1 ANIM

ANIM è un insieme di comandi messi a disposizione nei sistemi operativi di tipo UNIX per la produzione di animazioni, o anche solo “snapshot”, dell’esecuzione di un algoritmo.

Per generare la visualizzazione basta inserire all’interno del codice dell’algoritmo degli speciali comandi di output che produrranno il file di visualizzazione. In alternativa, il programma in esecuzione e gli strumenti di visualizzazione di ANIM possono essere collegati da un *pipe*, in modo da rendere la visualizzazione pressoché simultanea all’esecuzione dell’algoritmo.

ANIM permette la visualizzazione di programmi scritti in qualsiasi linguaggio.

La struttura di ANIM è molto semplice. Esso prevede quattro comandi per la visualizzazione di oggetti geometrici, che sono *text*, *line*, *box* e *circle*, e quattro comandi di controllo: *view*, *click*, *erase* e *clear*. Gli ultimi due eliminano, rispettivamente, un oggetto già disegnato in precedenza o tutti gli oggetti presenti nella visualizzazione. Il comando “view nome_finestra” seleziona una determinata finestra di visualizzazione, in modo tale che tutti i comandi successivi a quel determinato “view” vengano applicati alla finestra specificata. Infine il comando “click nome” serve a impostare all’interno del codice dei punti di controllo dai quali è possibile, ad esempio, dividere la visualizzazione in più visualizzazioni o generare uno snapshot.

Una mancanza del sistema ANIM è, forse, l’assenza di transizioni, ovvero di animazioni fluide, cosa che invece costituiva il punto di forza dei sistemi della famiglia TANGO (si ricordi il sopra citato *Path-Transition Paradigm*). In ANIM, infatti, come si può dedurre dai comandi presenti, un’animazione è prodotta cancellando l’oggetto del passo precedente e ridisegnando totalmente l’oggetto nella nuova posizione (o con nuovi colori o dimensioni).

1.4.2 Mocha

Mocha fu il primo sistema di visualizzazione di algoritmi distribuito. Mocha era quindi stato pensato come sistema di visualizzazione di algoritmi da utilizzare attraverso il Web.

Un client inviava al server Mocha una richiesta di visualizzazione, selezionando l’algoritmo da visualizzare ed altre opzioni come i valori di input, quindi il server Mocha eseguiva l’algoritmo e generava i comandi di visualizzazione, che sarebbero poi stati eseguiti da un Applet sulla macchina client [1] [6].

Il punto di forza di Mocha erano quindi rappresentato dal fatto che la pe-

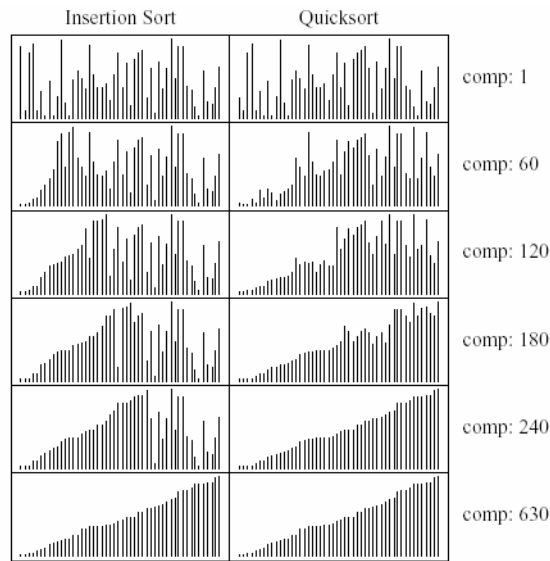


Figura 8: Snapshot presi a diverse iterazioni dell'InsertionSort e del Quicksort utilizzando gli strumenti ANIM per la visualizzazione grafica.

santezza dell'esecuzione dell'algoritmo gravava totalmente sul server ed esso poteva essere raggiungibile da qualsiasi macchina dotata di connessione alla rete Internet. Nonostante questo, il progetto Mocha non venne portato avanti e non superò mai la fase di prototipo.

1.4.3 *Leonardo*

Leonardo è un sistema per lo sviluppo e la visualizzazione di programmi in linguaggio C [19]. Essendo un ambiente di sviluppo è dotato di editor e di compilatore.

Una delle funzioni più interessanti di Leonardo era la "CPU virtuale invertibile": tutte le operazioni venivano eseguite su una CPU virtuale, che teneva traccia di tutte le operazioni eseguite e che poteva essere "riavvolta", annullando operazioni già eseguite. Questa funzione di Leonardo, combinata con la capacità di generare animazioni, risultava essere molto utile nella fase di debug di un programma.

Speciali comandi globali, detti *predicati*, del sistema Leonardo (scritti in linguaggio C) venivano messi a disposizione per produrre la visualizzazione. I predicati definiscono oggetti grafici, i loro parametri e la loro variabile corrispondente nel codice del programma stesso. La modifica del valore di una variabile causa quindi, in modo automatico, l'aggiornamento di tutti gli oggetti grafici ad

essa correlati.

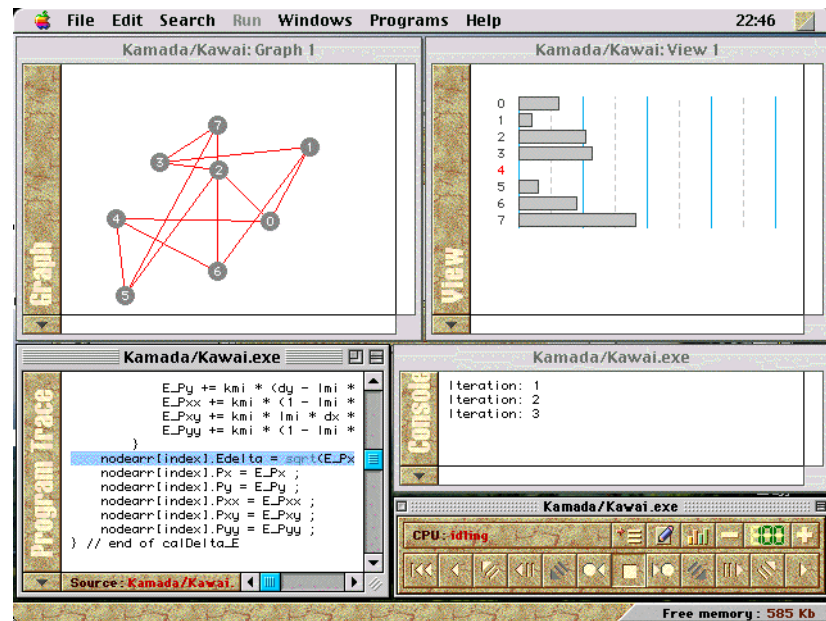


Figura 9: Sistema Leonardo durante l'esecuzione dell'algoritmo di tracciamento dei grafi di Kamada-Kawai.

1.4.4 GAWAIN

GAWAIN (*Geometric Algorithms Web-based AnImatioN*, ovvero *Animazione di algoritmi geometrici orientata al web*) era un sistema di visualizzazione di algoritmi sviluppato da Alejo Hausner presso Princeton nel 1998 con un concetto di base molto simile a quello del sistema Mocha: un Applet Java riceveva gli eventi generati dall'esecuzione dell'algoritmo sul server e produceva l'animazione sul client richiedente.

Come suggerisce il nome, GAWAIN era fortemente improntato alla visualizzazione di algoritmi di tipo geometrico, anche se poteva essere utilizzato come sistema di simulazione *general purpose* su altri tipi di algoritmo [5].

GAWAIN forniva, inoltre, all'utente un generatore di input, come nel sistema Balsa II, o anche la possibilità di inserire manualmente i valori dell'input dell'algoritmo da simulare.

La visualizzazione dell'algoritmo poteva essere eseguita sia in avanti che a ritroso e a differenti velocità (la velocità di visualizzazione viene misurata in eventi al secondo). Inoltre veniva supportata anche la visualizzazione di oggetti 3D, anche se le librerie utilizzate non erano perfettamente supportate. Erano supportati invece vari tipi di zoom, come in XTANGO, e la possibilità di im-

postare un diverso numero di eventi per ogni step, in modo che ad ogni passo della visualizzazione venissero saltati in modo automatico eventi ritenuti meno importanti.

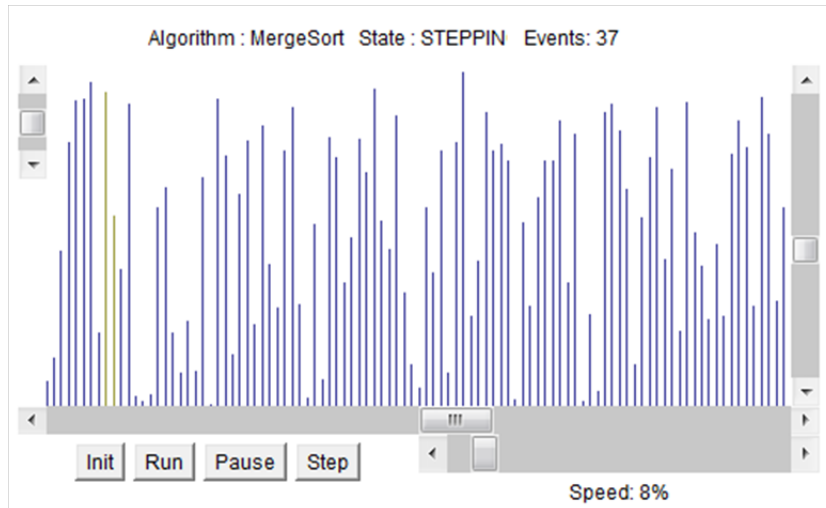


Figura 10: L'esecuzione dell'algoritmo del MergeSort in ambiente GAWAIN.

1.4.5 Pavane

Il sistema Pavane si discostava dagli altri sistemi di visualizzazione di algoritmi finora analizzati in quanto esso era basato su programmazione dichiarativa e non imperativa. In Pavane l'esecuzione del codice dell'algoritmo non produceva il codice relativo alla visualizzazione tramite un compilatore: era presente, infatti, un interprete che, durante l'esecuzione dell'algoritmo, riconosceva le variabili interessate alla visualizzazione [11].

Il programmatore che volesse visualizzare un algoritmo utilizzando Pavane doveva definire delle apposite formule logiche che definissero quando un determinato oggetto doveva essere visualizzato sullo schermo.

Quindi l'idea principale di Pavane era quella di visualizzare lo stato interno del programma sotto forma di immagini: ogni volta che un determinato stato veniva raggiunto, l'animazione veniva aggiornata.

Il gestore grafico di Pavane osservava, all'inizio della simulazione, lo stato delle variabili interessate per la visualizzazione. Ogni volta, poi, che all'interno del codice veniva richiamata la funzione *VisualUpdate()* (che è l'unico comando imperativo presente), esso comparava l'ultimo stato analizzato con quello corrente ed aggiornava l'animazione di conseguenza.

Per avere una simulazione completamente dichiarativa, era necessario eseguire

il programma su una macchina virtuale o utilizzare un qualche tipo di segnale che indicasse quando lo stato di una variabile osservata era cambiato. Questo approccio risultava essere molto costoso, per questo veniva utilizzata la funzione *VisualUpdate()*, che rendeva quindi Pavane un sistema ibrido: apposite formule logiche avevano il compito di controllare quali aspetti dello stato del programma erano cambiati rispetto all'ultimo controllo, ma era il programmatore stesso che decideva, tramite l'inserzione dei comandi *VisualUpdate()*, quando erano da effettuare tali controlli sullo stato del programma.

Infine Pavane supportava anche le visualizzazioni 3D, fornendo al programmatore maggiori possibilità di personalizzazione e miglioramento delle visualizzazioni.

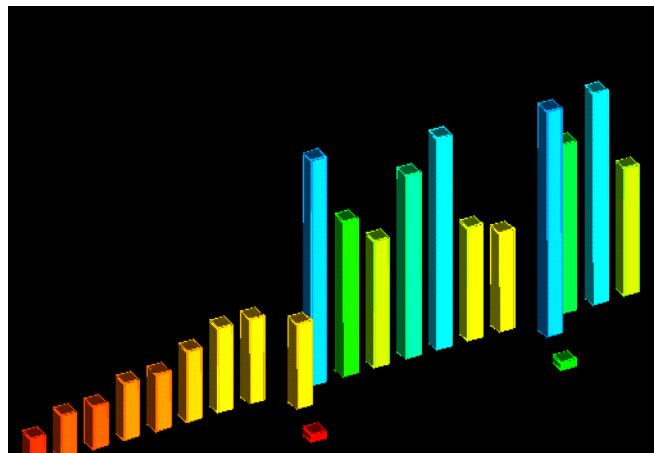


Figura 11: Visualizzazione a colori e 3D dell'esecuzione dell'algoritmo del QuickSort in Pavane.

1.4.6 CATAI

CATAI (*Concurrent Algorithm and data Types Animation over the Internet*, ovvero *Animazione su Internet di Algoritmi Concorrenti e Tipi di dato*) era un sistema di animazione che visualizzava programmi scritti in linguaggio C++ utilizzando una struttura distribuita e quindi permettendo l'esecuzione di algoritmi concorrenti e la visualizzazione da parte di più utenti di una stessa visualizzazione [6]. Le animazioni erano quindi visualizzate in un client Java.

CATAI offriva numerose possibilità d'interazione come, ad esempio, la possibilità di inserimento manuale dell'input tramite un'apposita interfaccia grafica. Inoltre CATAI implementava le visualizzazioni ad un alto livello di aderenza tra animazione e codice dell'algoritmo. Normalmente, infatti, negli altri sistemi di visualizzazione di algoritmi, si utilizzano tecniche di visualizzazione orientate agli eventi, ovvero in cui era il programmatore che doveva inserire nel codice

dell'algoritmo, in concomitanza con parti interessanti del codice, le funzioni grafiche che dovevano generare la visualizzazione. In CATAI invece la generazione dell'animazione veniva fatta in automatico, in modo da evitare possibili errori da parte del programmatore e da garantire una più stretta aderenza tra esecuzione e visualizzazione dell'algoritmo. Questo veniva fatto da CATAI utilizzando delle specifiche strutture dati che incorporavano il collegamento con la parte grafica, in modo che ogni modifica alla struttura dati stessa producesse in modo automatico l'aggiornamento corrispettivo nell'animazione.

1.5 SISTEMI DI VISUALIZZAZIONE DI ALGORITMI RECENTI

Fino ad ora abbiamo parlato dei sistemi di visualizzazione di algoritmi che hanno fatto la storia in questo settore. Vedremo adesso quali sono i sistemi di visualizzazione di algoritmi più recenti e promettenti.

1.5.1 JHAVÉ

JHAVÉ (*Java-Hosted Algorithm Visualization Environment*, ovvero *Ambiente di Visualizzazione di Algoritmi implementato in Java*) è un sistema di visualizzazione di algoritmi implementato in Java che realizza il modello architetturale del client-server. Il sistema è ottimizzato quindi per la visualizzazione via Web.

Il modello client-server di JHAVÉ funziona con una macchina client, che rappresenta l'utente, che invia al server la richiesta di visualizzazione dopo aver indicato l'algoritmo da visualizzare, i valori di input ed i valori dei parametri specifici dell'algoritmo (se presenti); la macchina server, quindi, esegue l'algoritmo con le impostazioni richieste ed invia al client, come risposta, lo script della visualizzazione che il client potrà mostrare.

JHAVÉ è fortemente improntato all'aspetto pedagogico della visualizzazione di algoritmi: infatti, oltre a fornire un'alta interattività tramite l'inserimento manuale degli input, un generatore di input o la gestione della velocità e della direzione (avanti o a ritroso) della visualizzazione, esso permette al docente di inserire, in punti specifici dell'esecuzione dell'algoritmo, le cosiddette *stop-and-think questions*, ovvero domande non estremamente complesse riguardanti il funzionamento dell'algoritmo in quel preciso momento.

Un'utile funzione di JHAVÉ è quella di raggruppare gli algoritmi, disponibili all'interno del suo database, secondo il tipo dei dati di input. È infatti ragionevole pensare che, dopo aver eseguito e visualizzato un algoritmo con un determinato input set, si possa voler eseguire e visualizzare, a scopi comparativi, un algoritmo diverso sullo stesso data set (un esempio possono essere i vari algoritmi di ordinamento).

Infine un punto di forza di JHAVE è la portabilità: i programmi da visualizzare possono, infatti, essere scritti in qualsiasi linguaggio di programmazione (purché essi richi amino le specifiche funzioni per la produzione degli script di visualizzazione, utilizzando o il motore grafico di SAMBA sviluppato da Stasko, o il motore GAIGS sviluppato da Naps). Una volta inviato il programma da visualizzare, esso verrà eseguito dall'applicazione Java sul server, indipendentemente dal linguaggio di programmazione utilizzato.

Un docente può quindi utilizzare gli algoritmi scritti in qualsiasi linguaggio egli desideri, senza essere costretto ad utilizzare Java.

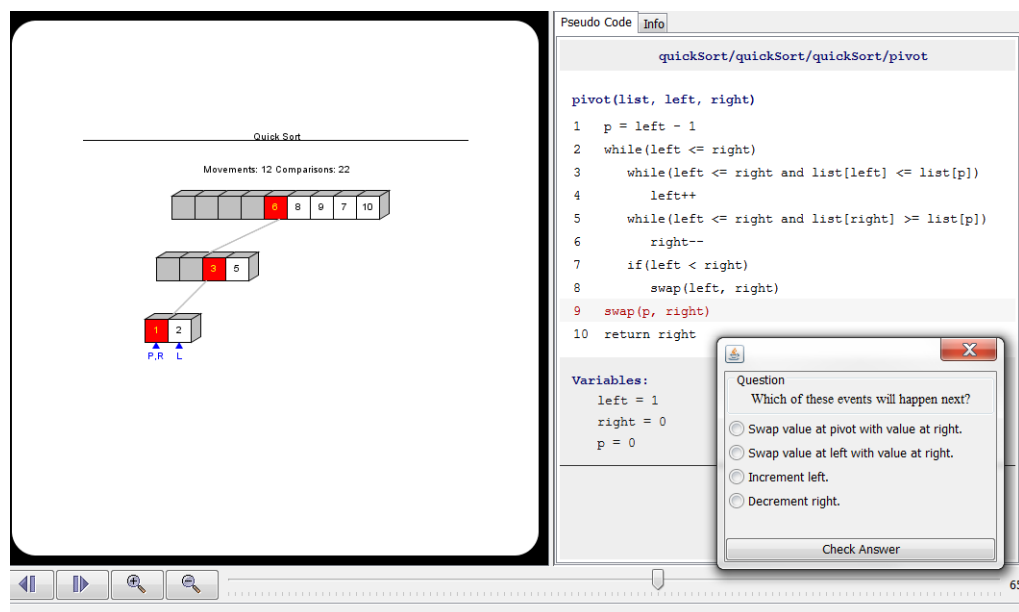


Figura 12: Un passo dell'esecuzione dell'algoritmo del QuickSort su JHAVE: sulla destra si possono notare lo pseudocodice, i valori degli indici utilizzati nell'algoritmo e una domanda sul prossimo passo.

1.5.2 Trakla

Il sistema Trakla è un sistema, sviluppato presso la Helsinki University of Technology, che permette agli studenti di risolvere svariati esercizi nel campo dell'algoritmica e di ottenere una risposta in modo automatico. Viene messo a disposizione, oltre al sistema di generazione e correzione degli esercizi, anche un editor grafico scritto in Java, denominato *TraklaEdit*, che permette di interagire con le strutture dati presenti nell'algoritmo in modo diretto, ad esempio trascinando con il mouse l'elemento di un vettore nella giusta posizione. Il sistema Trakla, assieme a *TraklaEdit*, può essere usato anche tramite una pagina Web, per permettere a chiunque di accedere al servizio ed ottenere le correzioni agli

esercizi. L'intero sistema operante sul Web viene detto *WWW-Trakla*.

Il sistema è dotato quindi di un server, detto *Trakla-server*. Inizialmente si poteva comunicare (inviare esercizi, ricevere correzioni, ecc.) soltanto attraverso una mail-box presente sul server. Più recentemente invece è stato implementato il sistema *WWW-Trakla* sopra citato che, assieme al *TraklaEdit*, risulta molto più comodo e immediato.

Un utente che utilizzi il sistema *Trakla* può scegliere di registrarsi sul server in modo da avere, da qualsiasi posto egli effettui il login, i propri esercizi e correzioni salvati ed una graduatoria con le votazioni precedenti.

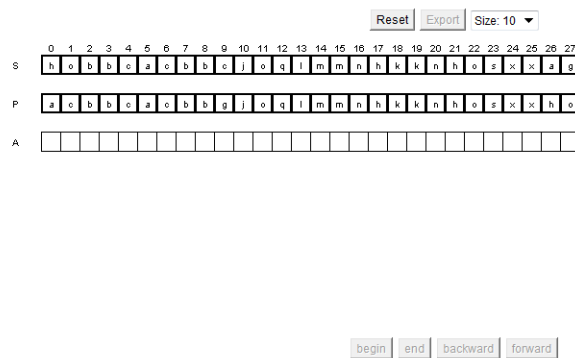


Figura 13: Un esercizio in *Trakla* in cui si deve determinare il perno nell'algoritmo del QuickSort.

1.5.3 *OpenDSA*

OpenDSA (dove la sigla *DSA* sta per *Data Structure and Algorithms*, ovvero *Algoritmi e Strutture Dati*) è un progetto finalizzato a fornire agli studenti importanti risorse per lo studio e la comprensione di algoritmi e strutture dati.

Il progetto *OpenDSA* viene anche definito con il termine *Active-eBook*, ovvero "libro digitale interattivo". Infatti *OpenDSA* punta a creare un percorso disciplinare semplice e lineare, improntato allo studio di algoritmi e strutture dati, fornendo spiegazioni e definizioni, in formato testuale, ma anche esempi ed esercizi, in formato grafico e interattivo.

Questo corso virtuale è quindi suddiviso in moduli, come ad esempio "Algoritmi di ordinamento" oppure "Alberi di ricerca": in ogni modulo saranno presenti diverse pagine di spiegazioni riguardanti l'argomento del modulo, esempi sugli algoritmi trattati (con la possibilità di personalizzare i valori di input e i parametri specifici dell'algoritmo) ed infine una grande quantità di esercizi, con cui lo studente potrà mettersi alla prova e verificare le proprie conoscenze, ottenendo in maniera immediata ed automatica una valutazione

sulla risoluzione dell'esercizio.

OpenDSA è, come suggerisce il prefisso *Open*, un progetto Open Source, nel senso che non soltanto i corsi virtuali messi a disposizione sono utilizzabili da chiunque, ma anche nel senso che il codice sorgente è visualizzabile e modificabile da chiunque voglia farlo: un professore può, ad esempio, creare i propri moduli, modificare le spiegazioni o aggiungere algoritmi o esercizi.

Inoltre viene messa a disposizione una particolare infrastruttura per permettere la registrazione di utenti al corso virtuale in modo che sia essi che un eventuale docente, possano seguire e monitorare i propri progressi durante lo studio dei moduli.

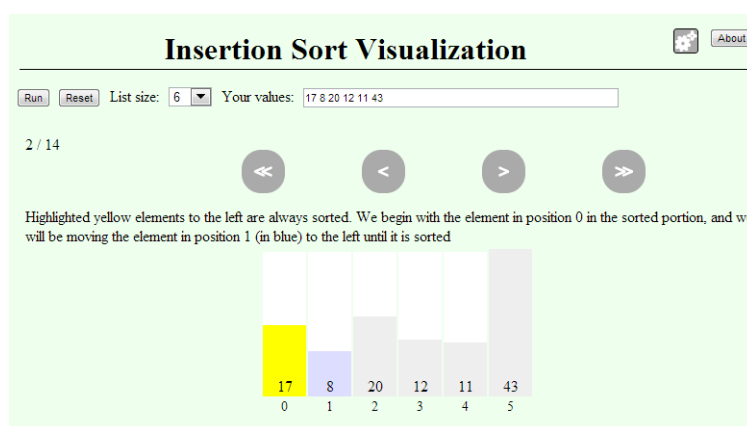


Figura 14: Un passo dell'esecuzione dell'algoritmo dell'InsertionSort con OpenDSA.

1.6 TASSONOMIA DEI SISTEMI DI VISUALIZZAZIONE DI ALGORITMI

La tassonomia è la disciplina che si occupa della classificazione gerarchica, ed eventualmente della nomenclatura, di elementi appartenenti ad uno stesso ambito. Una tassonomia risulta quindi molto utile per la comprensione di un ambito, o gruppo di elementi, in quanto tramite la classificazione strutturata di quest'ultimi sarà possibile realizzare analisi e comparazioni di questi elementi. Una tassonomia è inoltre utile a prevedere in quali direzioni avverranno (o dovrebbero avvenire) nuove scoperte. Infine una tassonomia dev'essere strutturata in maniera gerarchica, ovvero suddividendo gli elementi in categorie, sotto-categorie, sotto-sotto-categorie, ecc., e in modo che eventuali nuove scoperte possano essere incluse nella tassonomia senza dover cambiare o sconvolgere radicalmente l'impostazione precedente [3].

Proponiamo in questa Sezione una tassonomia per i sistemi di visualizzazione di algoritmi, indicando le categorie principali in cui essi possono essere suddivisi

e alcune sotto-categorie minori. Inoltre indicheremo, per ogni categoria, quali dei sistemi visti all'interno di questo Capitolo vi appartiene o meno oppure un valore approssimativo dell'efficacia di ogni sistema in quell'area.

Si osservi che, dal momento che ogni sistema è un insieme di più caratteristiche, non è possibile inserire in maniera assoluta un sistema in una specifica categoria (come solitamente viene fatto per gli esseri viventi nelle tassonomie biologiche).

Come si può osservare dalla Figura 15, sono state definite cinque catego-

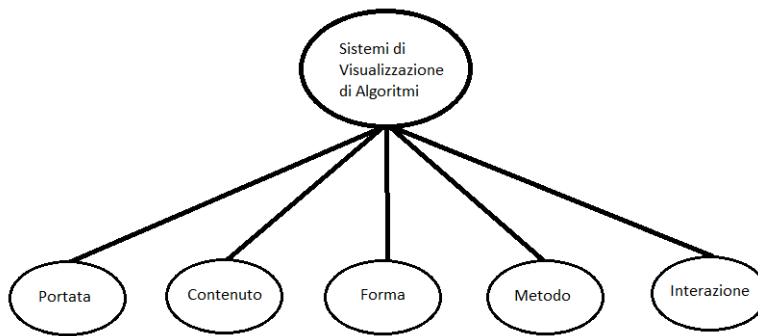


Figura 15: Il primo livello gerarchico della tassonomia dei sistemi di visualizzazione di algoritmi.

rie principali in cui articolare la classificazione dei sistemi di visualizzazione di algoritmi. Esse sono:

- **Portata:**
qual è la gamma di programmi che il sistema può prendere in input per la visualizzazione?
- **Contenuto:**
quale sottoinsieme delle informazioni gestite dal software sono visualizzate dal sistema?
- **Forma:**
quali sono le caratteristiche delle visualizzazioni?
- **Metodo:**
come viene specificata la visualizzazione?
- **Interazione:**
come interagisce l'utente con il sistema di visualizzazione di algoritmi?

1.6.1 Portata

La Portata risponde alla domanda “qual è la gamma di programmi che il sistema può prendere in input per la visualizzazione?”, ovvero si occupa di classificare i sistemi di visualizzazione di algoritmi in base alla *generalità* e alla *scalabilità* dei programmi passati come input per la visualizzazione.

Vediamo di seguito a quali domande risponde ogni sottocategoria appartenente

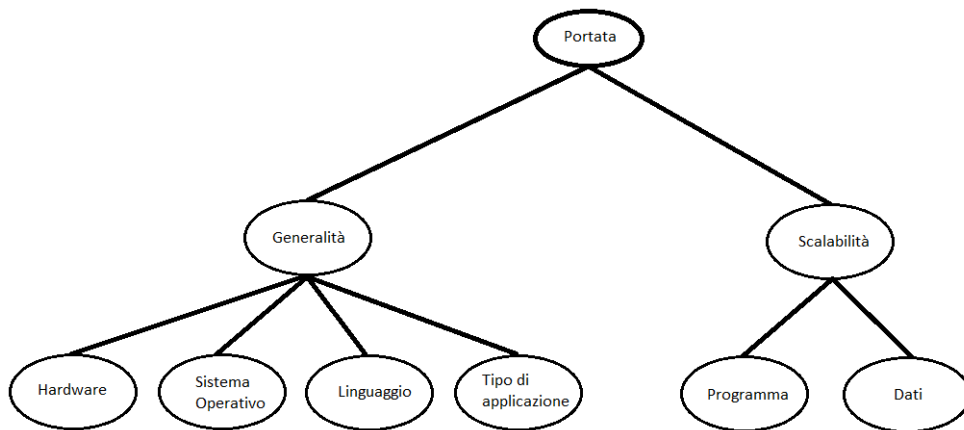


Figura 16: La struttura tassonomica della categoria della Portata, indicandone alcune sottocategorie.

alla Portata:

- Generalità: *il sistema generalizza sulla gamma di programmi visualizzabili o accetta soltanto un sottoinsieme ristretto di essi?*
 - Hardware: *che hardware è richiesto per utilizzare il sistema?*
 - Sistema Operativo: *che Sistema Operativo è richiesto per utilizzare il sistema?*
 - Linguaggio: *in che linguaggio di programmazione è necessario scrivere i programmi da visualizzare?*
 - Tipo di applicazione: *quali sono le restrizioni sul tipo di programma che può essere visualizzato?*
- Scalabilità: *qual è il grado di adattamento del sistema per esempi di grandi dimensioni?*
 - Programma: *qual è il più grande programma visualizzabile?*
 - Dati: *qual è il più grande insieme di dati visualizzabile?*

I video presentati all'inizio del Capitolo, L6 e Sorting Out Sorting, risultano molto carenti per generalità in quanto, essendo video preregistrati, non è possibile personalizzare la visualizzazione su nuovi esempi. I primi sistemi BALSA erano stati progettati per essere eseguiti soltanto sulle macchine delle *electronic classroom* della Brown University. Il sistema ANIM è un insieme di strumenti di grafica dei sistemi UNIX, quindi non utilizzabile al di fuori di essi. I sistemi operativi più recenti, invece, operano tutti sul web, permettendo quindi un'esecuzione su qualsiasi coppia macchina/sistema operativo.

Anche se più o meno tutti i sistemi analizzati permettono la visualizzazione di qualsiasi programma (previamente scritto in un linguaggio comprensibile al sistema) è da osservarsi che alcuni sistemi sono specializzati in determinati tipi di applicazioni: i sistemi della famiglia BALSA, ad esempio, visualizzano molto bene gli algoritmi che utilizzano vettori o grafi, mentre GAWAIN è specializzato in algoritmi di tipo geometrico.

Per quanto riguarda il linguaggio di programmazione in cui è necessario scrivere l'algoritmo da visualizzare, tra i primi sistemi, ANIM divenne famoso proprio per la possibilità di poter scrivere i programmi da visualizzare in qualsiasi linguaggio di programmazione. Tutti i sistemi della famiglia BALSA e TANGO, invece, richiedevano l'utilizzo di un particolare linguaggio di programmazione, che spesso rendeva difficile il compito di implementare uno specifico algoritmo per ottenerne la visualizzazione. Anche altri sistemi indipendenti richiedevano l'utilizzo di un particolare linguaggio di programmazione, come ad esempio Leonardo, che supportava il C, o Pavane, che supportava soltanto la programmazione dichiarativa (contrariamente a tutti gli altri sistemi che supportavano la programmazione imperativa). Tra i sistemi più recenti si distacca JHAVÉ, che permette la scrittura dei programmi in un qualsiasi linguaggio di programmazione scelto dall'utente.

Tecnicamente tutti i sistemi analizzati sono in grado di gestire programmi ed insiemi di dati arbitrariamente grandi, anche se pochi di essi forniscono esempi in questo senso. Il video Sorting Out Sorting, ad esempio, mostra visualizzazioni che gestiscono grandi quantità di dati.

Si presti attenzione al fatto che la scalabilità, ed in particolare la scalabilità sui dati da gestire, non risponde alla domanda "quanto il sistema visualizza bene grandi quantità di dati?", ma si occupa soltanto di stabilire i limiti effettivi di visualizzazione per grandi quantità di dati gestiti.

1.6.2 *Contenuto*

La categoria del Contenuto si occupa di classificare i sistemi di visualizzazione di algoritmi in base a cosa essi visualizzino e a come questa visualizzazione venga generata, comparando, nella maggior parte dei casi, la visualizzazione

prodotta con l'algoritmo e i dati effettivi.

Vediamo nel dettaglio le sottocategorie appartenenti al Contenuto e come

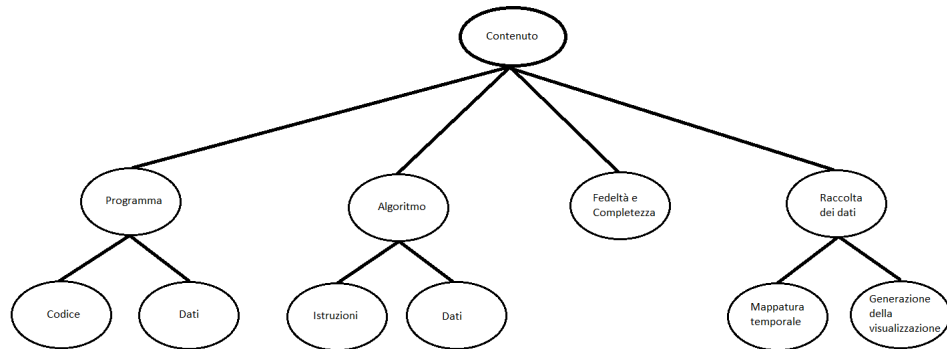


Figura 17: La struttura tassonomica della categoria del Contenuto, indicandone alcune sottocategorie.

vengono classificati in base a esse i sistemi studiati:

- **Programma:** *con che grado di aderenza il sistema visualizza il programma?*
 - **Codice:** *con che grado di aderenza il sistema visualizza le istruzioni del codice sorgente del programma?*
 - **Dati:** *con che grado di aderenza il sistema visualizza le strutture dati gestite dal programma?*
- **Algoritmo:** *con che grado di aderenza il sistema visualizza l'algoritmo?*
 - **Istruzioni:** *con che grado di aderenza il sistema visualizza le istruzioni dell'algoritmo?*
 - **Dati:** *con che grado di aderenza il sistema visualizza le strutture dati dell'algoritmo?*
- **Fedeltà e Completezza:** *le rappresentazioni grafiche rispecchiano l'attuale ed il completo comportamento del programma?*
- **Raccolta dei dati:** *i dati necessari per la visualizzazione vengono raccolti in fase di compilazione o di esecuzione?*
 - **Mappatura temporale:** *che corrispondenza sussiste tra il tempo del programma ed il tempo della visualizzazione?*
 - **Generazione della visualizzazione:** *la visualizzazione viene generata dopo aver eseguito il programma (batch) o durante la sua esecuzione?*

La visualizzazione del programma è attualmente implementata da pochi sistemi e più in particolare da nessuno di quelli elencati in questo Capitolo, che sono

fortemente orientati all'algoritmo, piuttosto che al programma. Ad ogni modo esistono sistemi che visualizzano, con un certo grado di aderenza, le istruzioni eseguite dal programma (per esteso o avvalendosi di uno pseudocodice), il flusso di controllo al momento dell'esecuzione (come ad esempio mostrando chiamate ricorsive) e le strutture dati gestite dal programma.

Come appena accennato, i sistemi studiati in questo Capitolo sono orientati alla visualizzazione di algoritmi, piuttosto che di programmi. Alcuni di essi, come Zeus, JHAVÉ o Leonardo forniscono il supporto allo pseudocodice, ovvero codice semplificato che rappresenta le effettive istruzioni dell'algoritmo che viene evidenziato a seconda dell'azione effettuata ad un certo passo della visualizzazione.

Per quanto riguarda la visualizzazione delle strutture dati, più o meno tutti i sistemi visualizzano le strutture dati principali di ogni algoritmo (o lasciano la scelta al programmatore), tralasciando in alcuni casi dettagli come indici o strutture dati di supporto o temporanee.

Come già accennato per la sottocategoria degli Algoritmi, non sempre vengono visualizzate tutte le strutture dati e le istruzioni dell'algoritmo. In generale i sistemi sviluppati per scopi ingegneristici, come debugging, risultano essere molto più fedeli e completi rispetto ai sistemi a stampo pedagogico, che solitamente propongono una versione semplificata e minimalista dell'algoritmo, mentre per il debugging o l'analisi delle prestazioni è necessario che siano visualizzati anche dettagli implementativi, come variabili di supporto o chiamate di sistema. In questo senso, i sistemi più recenti che abbiamo studiato, ovvero JHAVÉ, Trakla ed OpenDSA, assieme a tutta la famiglia TANGO, non presentano un'alta fedeltà e completezza rispetto all'algoritmo passato come input. Tra i sistemi studiati, POLKA è forse quello più fedele e completo, in quanto mostra dettagli come lo storico delle chiamate o il numero di thread attivi.

La mappatura temporale si differenzia in base al momento in cui vengono raccolti dati necessari alla visualizzazione e al tipo di animazione: se le informazioni raccolte sono corrispondenti ad istanti specifici dell'esecuzione e la visualizzazione generata è statica, allora la mappatura è "da statico a statico", mentre se la visualizzazione è animata, allora si parla di mappatura "da statico a dinamico"; se invece vengono raccolte informazioni in maniera continuativa entro certi intervalli temporali e la visualizzazione risulta essere statica, allora la mappatura è "da dinamico a statico" (come si può vedere in ANIM), mentre se l'animazione è dinamica, allora la mappatura è "da dinamico a dinamico", come per tutti i sistemi della famiglia TANGO.

Per quanto riguarda invece il tempo dedicato alla generazione della visualizzazione, la maggior parte dei sistemi analizzati, ed in particolare i sistemi distribuiti con architettura client-server, genera la visualizzazione *post-mortem*,

ovvero raccoglie informazioni durante l'esecuzione e, una volta terminata quest'ultima, si dedica alla produzione della visualizzazione. Nel sistema ANIM, la produzione della visualizzazione può essere effettuata in concomitanza con l'esecuzione dell'algoritmo tramite l'utilizzo di *pipes*. Tra i sistemi studiati in questo Capitolo, Pavane è forse l'unico che presenta una completa aderenza temporale tra l'esecuzione dell'algoritmo e la produzione della visualizzazione.

1.6.3 Forma

Ci occuperemo adesso di come classificare i sistemi operativi secondo la Forma, ovvero secondo le caratteristiche della visualizzazione prodotta dal sistema.

Vediamo di seguito le sottocategorie in cui si articola la Forma, per ottenere

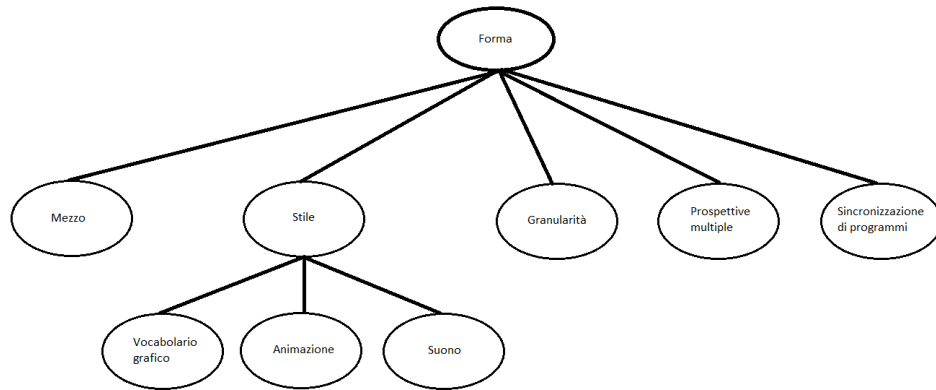


Figura 18: La struttura tassonomica della categoria della Forma, indicandone alcune sottocategorie.

un'ulteriore classificazione dei sistemi di visualizzazione di algoritmi:

- Mezzo: *qual è il mezzo di destinazione della visualizzazione?*
- Stile: *con che stile si presenta la visualizzazione?*
 - Vocabolario grafico: *quali elementi grafici compongono la visualizzazione?*
 - Animazione: *qual è il grado di animazione della visualizzazione?*
 - Suono: *quanto viene utilizzato il suono nella visualizzazione per trasmettere informazioni utili?*
- Granularità: *come vengono presentati dettagli a grana fine/larga?*
- Prospettive multiple: *come vengono gestite dal sistema prospettive sincronizzate di diversi aspetti del programma?*

- Sincronizzazione di programmi: *il sistema può generare visualizzazioni sincronizzate su più programmi contemporaneamente?*

I principali mezzi di destinazione per una visualizzazione sono carta, pellicola, video o monitor. I primi esempi, ovvero L6 e Sorting Out Sorting, erano stati progettati per essere visualizzati su pellicola, utilizzando un apposito proiettore. Gli altri sistemi sono principalmente indirizzati verso un generico monitor, anche se è da osservare che i primi sistemi studiati erano logicamente pensati per poter lavorare su monitor in bianco e nero, mentre soltanto sistemi più recenti, come Zeus, Pavane o Leonardo, furono sviluppati tenendo conto delle possibilità offerte dai più moderni monitor a colori.

Il vocabolario grafico definisce l'insieme di oggetti grafici che un sistema ha a disposizione per costruire la visualizzazione, ma anche aspetti come il colore o il 3D. In generale, un sistema può essere caratterizzato dalla dimensione del suo vocabolario grafico. Oggetti molto comuni sono figure geometriche elementari, come quadrati, linee, cerchi e rettangoli. I primi sistemi non utilizzavano una grafica a colori, fatta eccezione per il video Sorting Out Sorting: i primi sistemi in grado di produrre visualizzazioni a colori furono Leonardo, i sistemi della famiglia TANGO e GAWAIN. Ovviamente con l'evolversi della tecnologia, tutti i sistemi hanno implementato la grafica a colori, in quanto di gran lunga superiore, ai fini comunicativi, di quella in bianco e nero. Per quanto riguarda la grafica 3D troviamo, tra i sistemi che abbiamo studiato, Zeus, POLKA-3D, ANIM, Pavane, JHAVÉ, GAWAIN (quest'ultimo soltanto tramite l'utilizzo di apposite librerie 3D).

Riguardo alla distinzione tra animazione o visualizzazione statica, il primo sistema degno di nota in questo senso è TANGO che, implementando il cosiddetto *Path-transition paradigm*, riusciva a produrre animazioni fluide. Anche sistemi successivi, come Pavane, o i più recenti JHAVÉ e OpenDSA, generano animazioni fluide e molto gradevoli alla vista. È bene osservare che con il termine "visualizzazione statica" non si intende l'assenza di animazioni fluide, ma l'assenza totale di animazione, che si riduce alla semplice produzione di snapshot. In questo senso tutti i sistemi visti implementano una forma, seppure spesso molto rigida, di animazione (in sistemi come ANIM viene messa a disposizione la scelta tra la produzione di un'animazione, non fluida, o di snapshot).

Per quanto riguarda l'utilizzo di segnali sonori all'interno delle visualizzazioni, i primi sistemi implementavano soltanto dei semplici "beep" in particolari momenti dell'esecuzione, o non prevedevano affatto alcuna forma di output sonoro. Il primo sistema ad implementare input sonori personalizzati fu Zeus, della famiglia BALSA, che permetteva appunto di associare dei suoni a determinati eventi dell'esecuzione dell'algoritmo.

Visualizzare dettagli a grana fine dell'algoritmo è spesso importante in un

sistema di visualizzazione, specialmente se utilizzato ai fini del debugging. D'altra parte è molto importante anche che essi possano fornire una visione d'insieme del programma visualizzato, in modo da guadagnare in fatto di semplicità ed immediatezza: si parla in questo caso di rappresentazione a grana larga. Alcuni sistemi, come ad esempio POLKA, forniscono alcuni dettagli a grana fine dell'esecuzione del programma, ma i restanti sistemi sono fortemente indirizzati alla rappresentazione di informazioni a grana larga. In generale, non è possibile stabilire a priori se sia meglio utilizzare una rappresentazione a grana larga o fine: la scelta dipende strettamente dall'utilizzo che si vuol fare del sistema di visualizzazione.

Con "prospettive multiple" si intende la capacità di un sistema di poter visualizzare, in modo simultaneo e sincronizzato, diversi aspetti o parti di un programma, come la combinazione di prospettive a grana fine e a grana larga, o la rappresentazione di strutture dati dinamiche unita allo pseudocodice. Zeus fu uno dei primi ad implementare le prospettive multiple, seguito da POLKA, ANIM, Leonardo ed i più recenti JHAVÉ e OpenDSA.

Differentemente dalle Prospettive multiple, la Sincronizzazione di programmi valuta se e come un programma può visualizzare, contemporaneamente ed in maniera sincronizzata, più programmi, a scopi principalmente comparativi. Il video Sorting Out Sorting è il primissimo esempio di sincronizzazione di programmi: esso mostra in vari punti (vedi Figura 2d) delle vere e proprie "gare" tra algoritmi di ordinamento diversi operanti sullo stesso input set. Tra gli altri sistemi analizzati, forniscono la sincronizzazione di programmi i sistemi della famiglia Balsa, ANIM e CATAL.

1.6.4 Metodo

Il Metodo classifica i sistemi operativi in base a come viene creata la visualizzazione, ovvero in base alla sua *specifica* e alla *tecnica di connessione* utilizzata tra visualizzazione e codice sorgente.

Vediamo in dettaglio le sottocategorie presenti:

- Specifica della visualizzazione: *come viene specificata la visualizzazione?*
 - Intelligenza: *se la visualizzazione è automatica, quant'è avanzata l'intelligenza artificiale del sistema?*
 - Personalizzazione: *quanto è personalizzabile una visualizzazione dall'utente?*
- Tecnica di connessione: *come viene effettuata la connessione tra la visualizzazione ed il programma da visualizzare?*

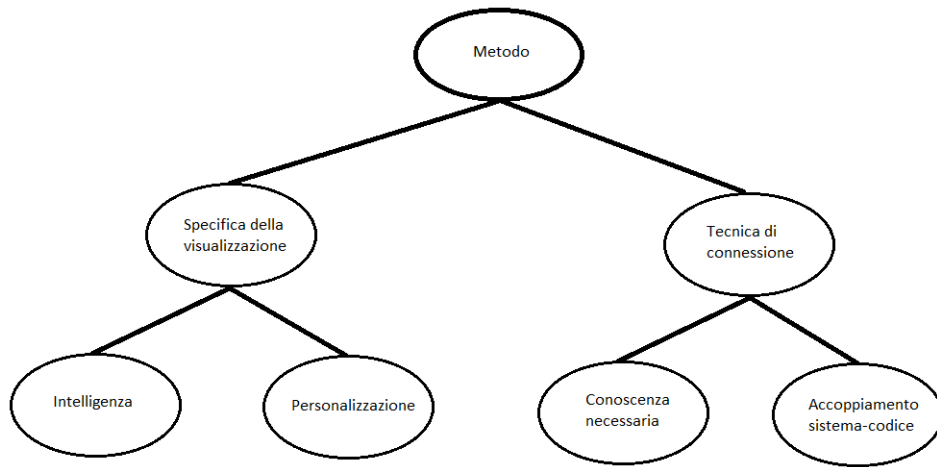


Figura 19: La struttura tassonomica della categoria del Metodo, indicandone alcune sottocategorie.

- Conoscenza necessaria: *se la visualizzazione non è completamente automatica, quali conoscenze di programmazione sono richieste all'utente per produrre la visualizzazione?*
- Accoppiamento sistema-codice: *quanto strettamente è accoppiato il sistema con il codice del programma?*

Per “specifica della visualizzazione” si intende il modo in cui viene specificata, a livello di codice, la visualizzazione. Essa può essere quindi: *scritta da zero*, ovvero scrivendo il codice della parte grafica utilizzando soltanto le primitive grafiche messe a disposizione dal sistema, *scritta utilizzando delle apposite librerie grafiche*, che standardizzano la grafica e ne semplificano la programmazione, oppure *automatica*, con il sistema dotato di un'intelligenza artificiale che decide quali strutture visualizzare e come. Solitamente sono quest'ultimo tipo di sistemi ad essere destinati al debugging, perché mostrano spesso dettagli a grana fine liberando il programmatore dall'onere di doversi occupare della parte grafica. Per quanto riguarda i sistemi studiati in questo capitolo, SAMBA è forse l'unico che mostra un certo grado di automazione nel processo di visualizzazione, seppur in modo non completo. I primi sistemi, come Balsa, Balsa II, TANGO, richiedevano una specifica scritta da zero, utilizzando le primitive grafiche del sistema, che risultavano spesso molto scomode. Già con i sistemi Zeus, per la famiglia Balsa, e XTANGO, per la famiglia TANGO, si videro implementate le prime librerie grafiche, presto seguite da librerie 3D.

In generale, si può dire che un sistema completamente automatico che visualizza tutto ciò che il programma gestisce e tutte le istruzioni che esegue, viene classificato come di “bassa intelligenza”; se invece il sistema automatico riesce da solo a riconoscere l'algoritmo e ad astrarre strutture dati di alto livello, allora si parla

di sistema ad “alta intelligenza”. Di conseguenza non avrebbe senso parlare di intelligenza in relazione ai sistemi trattati in questo Capitolo, in quanto tutti presentano una specifica della visualizzazione scritta da zero o si avvalgono di librerie grafiche.

La personalizzazione della visualizzazione è una caratteristica presente più o meno in tutti i sistemi, fatta eccezione per i due video di cui abbiamo parlato all’inizio di questo Capitolo, L6 e Sorting Out Sorting, che non permettevano alcun tipo di personalizzazione e vengono perciò detti *fissi*. I primi sistemi delle famiglie BALSA e TANGO proponevano una personalizzazione minima, che si riduceva alle dimensioni della finestra e allo zoom. Per sistemi successivi si ha un grado di personalizzazione maggiore, come la possibilità di scegliere colori, suoni, o anche di creare apposite interfacce tramite le quali l’utente possa inserire dati e manipolare la visualizzazione senza dover cambiare il codice.

La maggior parte delle tecniche di connessione richiede l’inserimento all’interno del codice sorgente di particolari istruzioni grafiche, che produrranno la visualizzazione: queste tecniche si dicono *invasive*. Esistono tecniche *non invasive* che prevedono l’utilizzo di particolari “sonde”, utilizzando un linguaggio dichiarativo, o di “monitor”. In questo senso, tutti i sistemi che abbiamo visto utilizzano delle tecniche di connessione invasive, in quanto richiedono la modifica, seppur minima, del codice sorgente. Leonardo rappresenta l’unica eccezione, dal momento che i comandi relativi alla visualizzazione vengono inseriti nel codice sorgente come commenti: è quindi possibile generare una visualizzazione con Leonardo e, successivamente, compilare lo stesso codice al di fuori dell’ambiente Leonardo, ottenendo gli stessi risultati (visualizzazione esclusa). Il sistema Pavana, essendo a stampo fortemente dichiarativo, mette a disposizione la possibilità di scegliere tra una tecnica di connessione invasiva, tramite l’utilizzo di funzioni *VisualUpdate()*, o una non invasiva.

Tutti i sistemi automatici non richiedono alcun tipo di conoscenze di programmazione, dal momento che essi possono produrre visualizzazioni senza che l’utente si preoccupi di aggiungere o modificare istruzioni. Ne consegue che tutti i sistemi visti richiedono la seppur minima conoscenza in fatto di programmazione: i sistemi che richiedono conoscenze più alte sono quelli che non implementano alcun tipo di libreria grafica, in quanto il programmatore dovrà specificare la visualizzazione da zero. Strumenti come librerie grafiche o, ancor meglio, editor grafici, semplificano ampiamente il lavoro di specifica della visualizzazione, richiedendo all’utente una minor conoscenza di programmazione. In ogni caso, quando si parla di sistemi non automatici, è necessaria anche un’intima comprensione del programma che si ha intenzione di visualizzare, dal momento che le istruzioni grafiche devono essere inserite in punti chiave dell’esecuzione del programma.

L’Accoppiamento sistema-codice ci dice infine quanto il sistema di visualizzazione ed il programma da visualizzare siano legati l’un l’altro. In sistemi come

BALSA, ad esempio, si ha un altissimo accoppiamento sistema-codice, in quanto è richiesto dal sistema che il programma venga eseguito e visualizzato all'interno dell'ambiente BALSA. Altri sistemi, come ANIM, non presentano alcun tipo di accoppiamento, in quanto leggono da un file generato dal programma le specifiche della visualizzazione.

1.6.5 Interazione

L'ultima categoria di questa tassonomia, l'Interazione, si occupa di classificare i sistemi di visualizzazione di algoritmi in base al grado di interazione che sussiste tra l'utente ed il sistema.

Come per le categoria precedenti, vediamo di seguito quali sottocategorie

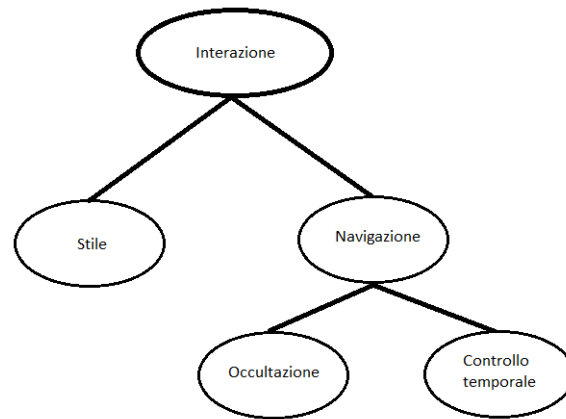


Figura 20: La struttura tassonomica della categoria dell'Interazione, indicandone alcune sottocategorie.

sono state individuate:

- *Stile: con che metodo l'utente dà istruzioni al sistema?*
- *Navigazione: come può navigare l'utente all'interno del sistema?*
 - *Occultazione: come può l'utente occultare parti superflue della visualizzazione?*
 - *Controllo temporale: come controlla l'utente lo scorrere dell'esecuzione del programma e della visualizzazione?*

Gli stili di interazione più comuni sono bottoni, menu o caselle di input testuale. Sotto questo punto di vista il sistema più evoluto è senz'altro JHAVÉ, in quanto mette a disposizione menu costruiti su misura per la scelta dell'input e dei parametri dell'algoritmo, ma anche risposte a scelta multipla per le domande che

possono essere inserite nella visualizzazione. Altri sistemi forniscono generatori di input o inserimento manuale dell'input, come si vede nei sistemi CATAI, GAWAIN, JHAVÉ e OpenDSA.

L'occultazione risulta essere molto importante quando il sistema fornisce così tante informazioni da pregiudicarne la chiarezza. È quindi essenziale poter escludere o nascondere dettagli della visualizzazione che non interessano e che non fanno altro che disturbare l'utente. In questo senso Leonardo, Zeus e JHAVÉ sono i sistemi che meglio implementano questo concetto, in quanto sono formati da finestre multiple, ognuna contenente una prospettiva diversa del programma. Infine, il controllo temporale indica se un sistema sia in grado di selezionare la *velocità* e la *direzione* di riproduzione della visualizzazione. Quasi tutti i sistemi analizzati forniscono il controllo della direzione della visualizzazione, permettendo di riprodurla in avanti o a ritroso. Il controllo sulla velocità, invece, non viene implementato in tutti i sistemi, in quanto non tutti permettono la riproduzione della visualizzazione come filmato, bensì come una successione di passi statici. Nel primo caso, tuttavia, è necessario avere a disposizione un controllo della velocità di riproduzione, per poter saltare parti della visualizzazione superflue e invece soffermarsi su parti interessanti. In questo senso, i sistemi che si avvalgono di controllo della velocità di riproduzione sono tutti i sistemi delle famiglie BALSA e TANGO ed i sistemi GAWAIN e JHAVÉ.

1.7 CONCLUSIONI

Concludiamo questo Capitolo dicendo che, tra i sistemi analizzati, i più recenti risultano essere molto promettenti e destinati ad interessanti sviluppi futuri. In particolare il sistema JHAVÉ risulta essere completo sotto tutti i punti di vista, fornendo strumenti utili sia per il debugging, che per l'apprendimento e la comprensione di algoritmi e strutture dati in ambiente scolastico. L'unica mancanza di questo sistema risiede, forse, in alcuni piccoli dettagli, come l'impossibilità di esportare la visualizzazione generata in formati video o HTML.

In generale, negli ultimi 30 anni si è potuto assistere ad una strabiliante evoluzione dei sistemi di visualizzazione di algoritmi, che si stanno sempre più affermando sia in ambito scolastico che lavorativo, specialmente grazie all'avvento di Internet, che ha permesso la nascita di sistemi più evoluti ed oggettivamente superiori rispetto ai loro antenati.

HTML5

HTML5 è un *linguaggio di markup* utilizzato nella strutturazione delle pagine *web*. Esso rappresenta la quinta versione dello standard *HTML* (un acronimo che sta per *HyperText Markup Language*), creato nel 1990 e la cui quarta versione, *HTML4*, venne standardizzata nel 1997.

2.1 LA NASCITA DI HTML5

HTML5 nasce da una scissione del W3C (World Wide Web Consortium) avvenuta dopo un convegno del 2004, durante il quale, per una manciata di voti, prevalse la linea orientata alle specifiche XHTML2. Dopo questo episodio, Ian Hickson fondò il gruppo di ricerca indipendente WHAT (*Web Hypertext Application Technology*, ovvero *Tecnologie per le Applicazioni orientate ad HTML*), il cui principale obiettivo era quello di garantire uno sviluppo del Web orientato più alle applicazioni che ai documenti [10].

Infatti HTML era stato sviluppato, inizialmente, per la stesura di semplici documenti testuali collegati tra loro tramite link. Questo era dovuto principalmente al fatto che durante la sua stesura (anni '90) le connessioni Internet erano ancora molto lente e le tecnologie per la produzione di immagini o video erano ancora piuttosto arretrate e costose. Era quindi raro vedere applicazioni sul Web, che risultavano a quei tempi costose ed esigenti in termini di banda. Tuttavia negli anni successivi le tecnologie subirono una brusca accelerazione, aprendo nuove possibilità di navigazione all'utente medio. Per questo motivo, si rese necessario sviluppare una specifica per le pagine Web che definisse una struttura solida e comoda per l'implementazione di applicazioni e servizi multimediali.

WHAT aveva quindi lo specifico intento di contrastare la linea di pensiero adottata dal W3C, in particolare riguardo all'abbandono delle specifiche HTML in favore di tecnologie orientate ad XML (*eXtensible Markup Language*).

2.1.1 XHTML2

XHTML2 (*eXtensible HyperText Markup Language*) è la seconda versione della specifica XHTML. Essa è formata dalla sintassi di HTML, costretto nelle regole ferree di XML, come l'obbligo della chiusura di tutti i tag. In questo modo, oltre ad utilizzare un linguaggio molto più rigido, risultava che il parser XML che

analizzava la pagina si sarebbe bloccato al primo errore riscontrato, annullando di fatto la visualizzazione dell'intera pagina (contrariamente a quanto accadeva con le specifiche HTML).

Inoltre, non veniva offerto alcun tipo di retrocompatibilità con la precedente versione 1.1.

Dovettero passare cinque anni dalla scissione del W3C prima che, finalmente, nel Luglio 2009 Tim Berners-Lee (inventore del World Wide Web e direttore del W3C) annullasse il progetto XHTML2, preferendo la strada intrapresa dal WHAT. Venne quindi formato un gruppo di ricerca unendo membri del W3C e del WHAT, sotto la guida di Ian Hickson, dal quale prese vita la specifica HTML5.

2.2 DA HTML4 AD HTML5

HTML5 rappresenta quindi un sostanziale passo in avanti rispetto alla versione precedente, HTML4, che si sta lentamente avviando verso la sua fine.

Al momento le specifiche HTML5 sono ancora in fase di studio e di sviluppo, ma ci sono buone ragioni per credere venga riconosciuto come standard ufficiale dal W3C già nel 2014, mentre la versione HTML5.1 dovrà attendere fino al 2016 affinché venga riconosciuta come standard.

Come già accennato nella Sezione precedente, una delle ragioni per cui è nato HTML5 è per avere pagine Web orientate ad applicazioni multimediali, piuttosto che a semplici documenti testuali. Fino ad HTML4, infatti, era necessario ricorrere, per poter eseguire o visualizzare applicazioni all'interno di una pagina Web, di appositi plug-in (ovvero software aggiuntivi di dimensioni ragionate) che, unitamente al browser di navigazione, permettevano la corretta visualizzazione di tutti i contenuti della pagina. L'esempio più famoso di questi plug-in è sicuramente Adobe Flash, che permette sia di creare animazioni interattive che di riprodurre streaming (cioè flussi) audio/video.

Con l'avvento di HTML5, invece, non ha più molto senso parlare di plug-in in quanto la maggior parte delle funzioni offerte da questi software aggiuntivi viene adesso supportata nativamente dalla stessa specifica HTML5.

Le principali novità introdotte dalla specifica HTML5 rispetto alla precedente versione sono:

- controllo di flussi multimediali (audio/video);
- produzione di grafica 2D e 3D in tempo reale;
- controllo dello storico di navigazione;
- marcatura unica e semplificata del *DOCTYPE*;

- aggiunta di nuovi tag;
- eliminazione di tag obsoleti;
- scrittura semplificata di formule matematiche;
- gestione degli errori;
- accesso ad applicazioni Web anche in assenza di connessione Internet;
- API per implementare funzioni come il drag-and-drop o la possibilità di modificare il testo;
- controllo su informazioni generate dall'utente tramite device come un microfono o una webcam.

In HTML5 lo streaming di audio e video è implementato nativamente grazie ai nuovi tag `<audio>` e `<video>`. Con questi nuovi tag risulta semplicissimo inserire audio e video nella pagina Web: basta specificare il file da riprodurre (in un qualsiasi formato audio o video), mentre altri campi, come il tipo di file o il codec necessario per la riproduzione, sono opzionali.

La produzione di grafica 2D (vettoriale oppure bitmap) e 3D in tempo reale è un'altra importante novità di HTML5. Il tag corrispondente alla grafica bitmap (o raster) è `<canvas>`, ma parleremo di Canvas in modo più approfondito nella prossima sezione, dato che esso costituisce il cuore del lavoro di espansione di AlViE al Web. Per la costruzione di figure 2D in grafica vettoriale viene invece utilizzata la tecnologia SVG (*Scalable Vector Graphics*, ovvero *Grafica Vettoriale Scalabile*), ovvero una particolare specifica XML che permette di definire forme, linee e curve per la costruzione di immagini perfettamente scalabili senza perdita di definizione.

Per quanto riguarda la grafica 3D, non esiste un vero e proprio supporto esplicito per la costruzione di oggetti 3D. Tuttavia, è stato mostrato [4] che risulta essere piuttosto semplice definire un motore grafico 3D utilizzando lo stesso tag `<canvas>` della grafica in 2D.

Il DOCTYPE è uno standard (viene messo come tag all'inizio del codice della pagina Web) che in sostanza dice al browser di navigazione a quale tipo di documento appartiene la pagina che si vuole visualizzare. Fino alla versione 4 di HTML, i DOCTYPE erano molto diversificati e composti da molti elementi. Un esempio di vecchio DOCTYPE può essere il seguente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3c.org/TR/html4/strict.dtd">.
```

In HTML5 il discorso del DOCTYPE viene ampiamente semplificato, lasciando che sia in gran parte il browser ad accorgersi da solo di che tipo di documento

si tratti. A titolo di esempio, il precedente DOCTYPE può essere sostituito, in HTML5, dal seguente tag:

```
<!DOCTYPE html>.
```

Parlando dei tag in generale, si nota subito che con l'arrivo di HTML5, molte cose sono cambiate: molti nuovi tag sono stati affiancati a quelli già conosciuti, mentre altri, che erano ormai deprecati, sono stati eliminati del tutto dalla nuova specifica. La maggior parte dei tag eliminati sono tag che sono stati sostituiti da nuove versioni più comode o che non permettevano di avere un'adeguata pulizia del codice HTML. Alcuni esempi possono essere i tag `` o `<centre>`, che vengono completamente sostituiti dall'utilizzo di CSS (*Cascading Style Sheet*, ovvero *Fogli di Stile a Cascata*), come anche i tag `<frame>`, `<frameset>` o `<strike>`.

Tra i nuovi tag troviamo ovviamente `<audio>` e `<video>`, per la riproduzione di file multimediali, il tag `<canvas>`, per la produzione di grafica 2D e 3D in tempo reale, i tag `<section>` e `<article>`, per la suddivisione della pagina in sezioni e sottosezioni in modo semplice ed immediato (facilitando anche l'uso dei CSS), ed altri tag come `<nav>`, `<header>`, `<footer>`, `<aside>`.

Come accennato nella Sezione precedente, parlando della nascita di HTML5, questo nuovo standard è il risultato degli studi del gruppo di ricerca che ha visto lavorare assieme membri del W3C e membri del WHAT. HTML5 presenta quindi due diverse facce: una che mostra la sua natura improntata alle tecnologie e alla semantica di XML, l'altra che indica l'orientamento di HTML5 verso le applicazioni ed il campo dei multimedia. Come tutte le specifiche basate su XML, prevede quindi regole ferree per quanto concerne la sintassi. Allo stesso tempo, HTML5 rimane una specifica molto flessibile, permettendo di definire le proprie regole di parsing (ovvero l'analisi ed il riconoscimento degli elementi della pagina) e di gestione degli errori: in questo modo si avrà la certezza che, in caso di errori di sintassi, tutti i browser producano lo stesso risultato sulla pagina, cosa che non era affatto garantita in HTML4, dove non esistevano regole precise da applicare in caso di errore ed ogni browser poteva implementare la gestione di tali errori in maniera diversa.

HTML5 fornisce anche il supporto alla scrittura di formule e simboli matematici grazie al linguaggio MathML (*Mathematical Markup Language*, ovvero *Linguaggio Matematico di Marcatura*). Questo linguaggio risulta essere molto semplice e conta poco più di 30 elementi. Una caratteristica interessante è che MathML non si occupa soltanto dello stile di una formula, cioè di come essa appare nella pagina, ma anche del suo significato matematico, cosicché una stessa formula possa essere rappresentata in diversi modi, o tramite diversi dispositivi, a discrezione dell'utente.

2.2.1 Canvas

Canvas (*tela*, in inglese) è un elemento di HTML5 per la produzione in tempo reale di figure 2D e la creazione di immagini bitmap. Canvas consiste in un'area della pagina, dotata di attributi altezza e larghezza, in cui si possono disegnare figure geometriche tramite l'invocazione di determinate funzioni in ambiente JavaScript.

A prima vista lo standard SVG, a cui abbiamo accennato brevemente prima, può sembrare superiore a Canvas: SVG, infatti, è una tecnologia per generare grafica vettoriale, il che significa che utilizzando SVG verrà memorizzato dal browser non soltanto l'immagine disegnata, ma anche "come" essa è stata disegnata. Più in particolare SVG definisce una serie di figure geometriche, linee e curve, facendo in modo che venga memorizzato come è stata creata ogni forma. In questo modo, tentando di ingrandire l'immagine, essa scalerà perfettamente, senza alcuna perdita di qualità. Questo è dovuto al fatto che, ricordando le forme originarie che hanno poi costruito l'immagine, se viene richiesta l'immagine in una scala diversa, il browser ricalcolerà nuovamente l'immagine partendo dalle forme che la definiscono (scalandone le dimensioni).

Canvas, al contrario, è uno strumento per la grafica bitmap (detta anche *raster*), il che significa che una volta generata l'immagine, viene "dimenticato" tramite quali forme primitive essa era stata costruita. Si avrà quindi, in caso di scalatura, una perdita di qualità, in quanto verrà ricordata soltanto l'immagine finale, intesa come matrice di pixel.

Questa principale mancanza di Canvas rispetto a SVG è compensata da una maggior semplicità del codice Canvas rispetto alla controparte SVG: l'esecuzione del codice Canvas, e la relativa renderizzazione dell'immagine, risultano essere estremamente veloci e molto leggeri in quanto ad occupazione di memoria [2]. Inoltre, l'implementazione in una pagina Web di grafica vettoriale tramite SVG, richiede un file .svg corrispondente, oltre al file .html della pagina stessa. Quindi più immagini distinte, o una serie di immagini, richiedono un file .svg ciascuna, mentre con Canvas tutto il codice relativo alla produzione dell'immagine può essere scritto nel codice della pagina, essendo basato su JavaScript.

Questi vantaggi, appena illustrati, di Canvas rispetto ad SVG, ci hanno condotti verso la scelta del primo dei due metodi per portare a termine l'obiettivo principale di AlViE4, ovvero l'esportazione di una visualizzazione in una pagina HTML. Come vedremo infatti nel Capitolo 4, per la generazione di visualizzazioni in HTML, verrà definito un apposito compilatore. Per questo motivo, dover scrivere su un unico file i comandi per la costruzione delle immagini della visualizzazione, è risultato essere molto comodo, senza contare il fatto che le proprietà di scalabilità offerte da SVG risultano essere pressoché inutili in questo tipo di progetto. Inoltre, con questa soluzione, si può esportare una visualizzazione completa in un'unica pagina HTML, che risulta sicuramente molto più

comoda per nel caso in cui un utente voglia utilizzare la stessa visualizzazione nel proprio sito Web.

UTILIZZO DEL CANVAS. Entriamo adesso nel dettaglio dei comandi messi a disposizione da Canvas, mostrando alcuni semplici esempi.

Innanzitutto dev'essere creato il Canvas stesso, ovvero l'area su cui andremo a disegnare tramite funzioni JavaScript. La creazione del canvas si ottiene semplicemente come:

```
<canvas id="canvas" width="200" height="200">  
  Se viene visualizzato questo messaggio, il browser non supporta Canvas!  
</canvas>
```

Il Canvas viene quindi creato semplicemente con l'apertura e la chiusura del tag corrispondente. Eventualmente, può essere inserito, come in quest'esempio, un messaggio all'interno del tag `<canvas>`, che verrà mostrato a video nel caso in cui Canvas non sia supportato dal browser scelto per la navigazione. L'attributo `id` serve a identificare, successivamente il canvas: il nome dell'id è del tutto arbitrario. I campi `width` e `height` rappresentano, rispettivamente, larghezza ed altezza della tela di disegno in pixel.

La mera creazione dell'oggetto Canvas non è, tuttavia, sufficiente per iniziare a disegnare. Per fare ciò, c'è bisogno di ottenere un oggetto, detto *contesto del disegno*, tramite le seguenti due linee di codice HTML in ambiente JavaScript:

```
var canvas = document.getElementById("canvas");  
var contesto = canvas.getContext("2d");
```

Possiamo, a questo punto, invocare le funzioni JavaScript di Canvas sul contesto appena definito e, quindi, iniziare di fatto a disegnare.

Vediamo, come primo esempio, il seguente codice:

```
contesto.beginPath();  
contesto.fillStyle = "FF0000";  
contesto.rect(75, 75, 50, 50);  
contesto.fill();  
contesto.endPath();
```

Si nota innanzitutto che tutti i comandi per il disegno, in Canvas, devono essere racchiusi tra `contesto.beginPath()` e `contesto.endPath()`. L'attributo `fillStyle` del contesto indica che colore utilizzare per riempire le figure geometriche che andremo a definire, utilizzando la codifica RGB (*Red Green Blue*, ovvero *Rosso Verde Blu*) esadecimale: in questo caso viene quindi selezionato il colore rosso. Con il comando `contesto.rect(75, 75, 50, 50)` si definisce, finalmente, la prima figura da disegnare: in questo caso si è definito un rettangolo, con l'angolo superiore sinistro (detto *origine*) nel punto (75, 75) e con larghezza ed altezza pari a 50 (ovvero, un quadrato). Si osservi che in Canvas l'origine delle coordinate viene identificata con l'angolo superiore sinistro: valori alti delle ascisse si trovano



Figura 21: Un quadrato rosso costruito utilizzando Canvas

sulla destra, mentre valori alti delle ordinate si trovano in basso al Canvas. Infine, il comando `contesto.fill()` indica che tutte le figure precedentemente definite (in questo caso, soltanto il quadrato) debbano essere riempite (*fill*, in inglese) del colore definito dall'attributo `fillStyle`, che come valore di default contiene il colore nero. Aggiungendo il codice appena descritto si ottiene l'immagine in Figura 21.

Proviamo adesso ad inserire le seguenti tre istruzioni tra i comandi `rect()` e `fill()` del codice precedente:

```
contesto.moveTo(75, 75);
contesto.lineTo(100, 50);
contesto.lineTo(125, 75);
```

L'istruzione `moveTo()` sposta il cursore del Canvas nelle coordinate indicate, senza però disegnare niente: un po' come muovere una matita sopra un foglio staccando la punta dalla superficie di disegno. Le due istruzioni `lineTo()`, invece, spostano il cursore verso la posizione indicata ma tracciando una linea sul Canvas. Quello che si ottiene, quindi, è di aver aggiunto un triangolo, sempre di colore rosso, sopra al quadrato di prima.

Facciamo un'ulteriore modifica, aggiungendo le seguenti due linee subito prima del comando `endPath()`:

```
contesto.lineWidth = 5;
contesto.stroke();
```

In questo modo indichiamo al Canvas di tracciare (e non riempire) le forme precedentemente definite, con una linea larga 5 pixel. Il colore delle linee è nero di default, ma può essere cambiato, così come per il colore di riempimento, impostando opportunamente il campo `strokeStyle`. Il risultato ottenuto da queste due modifiche è mostrato in Figura 22.

Aggiungiamo adesso la linea

```
contesto.fillText("Hello World!", 70, 145);
```



Figura 22: Una casetta rossa costruita utilizzando i comandi di Canvas



Figura 23: Una casetta rossa con messaggio costruita utilizzando i comandi di Canvas

subito dopo `beginPath()`, per ottenere la scritta “Hello World!” sotto alla figura precedente, come mostrato in Figura 23.

ALTRI COMANDI E ATTRIBUTI. Analizzeremo adesso, per concludere questo approfondimento su Canvas, altri importanti comandi grafici.

Un comando Canvas molto utilizzato è `contesto.clearRect(x, y, larghezza, altezza)` che ripulisce tutta l’area rettangolare del Canvas che ha origine in (x,y) e di larghezza ed altezza specificate. Questo comando risulta essere molto utile quando si deve visualizzare una successione di immagini, eventualmente con determinate immagini che compaiono al verificarsi di un certo evento, come il click su un bottone. Per il progetto AlViE4, questo comando è risultato essenziale: esso viene utilizzato all’inizio di ogni passo dell’algoritmo per ripulire l’intera tela del Canvas dalle strutture dati del passo precedente e poter quindi disegnare quelle al passo corrente.

L’attributo `font` del contesto grafico gestisce l’aspetto delle scritte disegnate sul Canvas. Quest’attributo è una stringa della forma

`[punti]pt [font] [STILE],`

dove `[punti]` rappresenta la grandezza dei caratteri, `[font]` indica quale font utilizzare e `[STILE]` definisce, appunto, lo stile di scrittura, come ad esempio corsivo o grassetto. Un esempio può quindi essere:

14pt Courier PLAIN.

È bene far presente che, per la gestione del colore dei testi, ci si rifà allo stesso attributo di riempimento, `fillStyle`, visto negli esempi precedenti.

Vediamo adesso nello specifico alcuni comandi particolari detti *modificatori*: `scale()`, `rotate()` e `translate()` e l'attributo `globalAlpha`.

Il primo comando citato, `contesto.scale(x, y)`, è utilizzato per scalare i disegni sulla tela del Canvas. I valori `x` ed `y` rappresentano i fattori di scalamento del Canvas rispetto agli assi delle ascisse e delle ordinate. Qualsiasi valore reale positivo può essere assegnato ad `x` e `y`: in particolare 1 indica di non scalare quell'asse, 0.5 indica il dimezzamento delle dimensioni e 2 il raddoppiamento.

Il comando `contesto.rotate(a)` serve a ruotare l'intero Canvas. L'unico parametro presente, `a`, indica l'angolo di rotazione in radianti.

Come si può intuire dal nome, il comando `contesto.translate(x, y)` permette di traslare tutta l'area del Canvas di `x` ed `y`, con `x` ed `y` che si riferiscono rispettivamente agli assi delle ascisse e delle ordinate.

Infine, l'attributo `globalAlpha` controlla la trasparenza del Canvas: esso assume valori reali da 0 a 1, con 0 che indica una trasparenza totale ed 1 una completa opacità del disegno (o assenza di trasparenza). Il valore di default per l'attributo `globalAlpha` è 1, quindi a meno che non venga assegnato esplicitamente un altro valore, i disegni in Canvas saranno inizialmente sempre opachi.

ALVIE

ALViE (*Algorithm Visualization Environment*) è un ambiente per lo sviluppo e la visualizzazione di algoritmi scritto in linguaggio Java.

ALViE è un sistema di visualizzazione di algoritmi *post-mortem* orientato agli eventi.

3.1 CARATTERISTICHE PRINCIPALI

Vedremo, in questa Sezione, quali sono le caratteristiche principali del sistema ALViE, cercando di collocare quest'ultimo nelle opportune categorie della tassonomia proposta all'interno del Capitolo 1.

I limiti di portata di ALViE sono piuttosto ampi: essendo sviluppato in ambiente Java, il sistema risulta essere multi-piattaforma e quindi eseguibile a prescindere dall'hardware o dal Sistema Operativo (purché sia installata una distribuzione Java aggiornata). Anche sui tipi di algoritmo visualizzabili non ci sono particolari restrizioni. Per quanto riguarda la scalabilità, ALViE non presenta problemi a gestire algoritmi o dati di grandi dimensioni, quindi i limiti del sistema coincidono con i limiti della macchina su cui viene eseguito.

L'unica restrizione presente è il linguaggio di programmazione: è infatti necessario che gli algoritmi da voler visualizzare siano scritti in linguaggio Java.

Com'è intuibile dal nome del sistema stesso, ALViE è orientato agli algoritmi, piuttosto che al programma. Come già accennato prima, ALViE è un sistema *post-mortem*, ovvero è un sistema che raccoglie dati durante l'esecuzione dell'algoritmo e soltanto dopo genera la visualizzazione a partire dall'insieme di questi dati. La mappatura temporale tra l'esecuzione e la visualizzazione è di tipo "da statico a statico", ovvero vengono generati degli snapshot, istantanee dell'esecuzione del programma prive di un qualsiasi tipo di animazione in fase di visualizzazione.

Il mezzo di destinazione principale di ALViE è un generico monitor a colori, di qualsiasi formato o risoluzione. Il vocabolario grafico di ALViE è composto dalle principali figure geometriche 2D: si hanno quadrati, rettangoli, cerchi, ovali, linee e frecce. Inoltre sono presenti oggetti di tipo testuale, come etichette, e lo pseudocodice, le cui linee vengono evidenziate con il progredire della visualizzazione. Come già accennato prima, ALViE non supporta le animazio-

ni, quindi il passaggio da un passo dell'algoritmo al seguente risulterà in un cambio netto tra l'immagine precedente e la nuova. Al momento, il sistema non supporta nemmeno l'utilizzo di suoni all'interno delle visualizzazioni. Tutte le visualizzazioni di ALViE sono a colori.

Per quanto riguarda la granularità dell'informazione visualizzata, ALViE visualizza prevalentemente informazioni a grana larga e strutture dati di alto livello, cercando quindi di astrarre il più possibile il programma per poter visualizzare l'algoritmo sottostante.

In ALViE, la specifica della visualizzazione è orientata agli eventi, il che significa che il processo di produzione di una visualizzazione non è automatico, ma richiede l'inserimento di particolari funzioni all'interno del codice sorgente dell'algoritmo. Più in particolare, ALViE ha bisogno di una classe, interna all'algoritmo, che definisca le strutture dati grafiche e gli eventi. Le strutture grafiche avranno una controparte all'interno dell'algoritmo e verranno poi utilizzate per aggiornare la visualizzazione. Gli eventi invece definiscono azioni o cambiamenti sulla visualizzazione, come un oggetto grafico che cambia di colore o di posizione.

Il livello di personalizzazione della visualizzazione è piuttosto alto: un qualsiasi utente con una conoscenza media del linguaggio Java può tranquillamente utilizzare le funzioni grafiche e le strutture dati messe a disposizione dal sistema per definire la propria visualizzazione o modificare visualizzazioni esistenti.

Essendo ALViE orientato agli eventi e non automatico, è richiesto l'utilizzo di una tecnica di connessione invasiva che prevede, quindi, l'inserimento di opportune invocazioni all'interno del codice sorgente originale dell'algoritmo.

ALViE mette a disposizione una serie di bottoni per l'interazione utente-sistema. Un bottone permette di modificare le impostazioni generali di ALViE, come ad esempio l'ubicazione della cartella contenente gli algoritmi da visualizzare. Con un altro bottone si può selezionare un algoritmo da eseguire e visualizzare: una lista di algoritmi disponibili permette di scegliere con un semplice click, quindi si dovrà scegliere, tramite delle apposite finestre, dove salvare la visualizzazione che sarà generata e da dove leggere l'input dell'algoritmo. Sono poi disponibili un bottone per caricare una visualizzazione preesistente ed uno per modificare le impostazioni di visualizzazione, come la grandezza del testo o il colore degli oggetti grafici. Si hanno quindi quattro bottoni di navigazione, che permettono di andare al passo precedente, al successivo, al primo o all'ultimo, e tre bottoni per lo zoom. Sono infine stati introdotti, nella quarta versione del sistema, altri due bottoni, per l'esportazione della visualizzazione in HTML5 e per la creazione manuale di un file di input, ma di queste due nuove funzionalità parleremo ampiamente più avanti.

3.1.1 Specifica delle visualizzazioni

Le visualizzazioni prodotte da ALViE vengono memorizzate sul disco come file XML.

L'intero file è quindi suddiviso in una serie di tag, spesso annidati tra loro.

Il tag più esterno, che racchiude tutta la visualizzazione, è `<algorithm name="NomeAlgoritmo">` e non presenta attributi significativi, se non il nome dell'algoritmo. All'interno del tag `<algorithm>` troviamo una serie di tag della forma `<step number="x">`, con x intero maggiore o uguale a 0, ognuno dei quali racchiude la specifica di un singolo passo di visualizzazione.

Ogni step è formato da uno o più tag della forma `<structure name="NomeStruttura" type="TipoStruttura">`, con il campo `type` che può assumere i seguenti valori: Array, Pseudocode, List, Queue, GeometricFigure, Graph o BinaryTree. Il numero delle strutture presenti ad ogni passo è arbitrario, anche se solitamente la struttura Pseudocode appare una ed una sola volta ad ogni passo.

Ogni struttura è composta da tre elementi: la struttura dati vera e propria, la sua controparte grafica ed un messaggio. Il messaggio serve per commentare il passo corrente dell'algoritmo e solitamente viene lasciato vuoto in tutte le strutture tranne una (per step). Della specifica della struttura dati vera e propria parleremo più avanti. Per quanto riguarda invece la struttura dati grafica, essa contiene informazioni sulle coordinate in cui visualizzare i vari elementi che compongono la struttura ed altre caratteristiche come altezza, larghezza, colore o forma. Nel tag esterno della struttura dati grafica vengono specificate le caratteristiche di default degli elementi che la compongono: gli elementi vengono quindi specificati all'interno del tag soltanto se hanno almeno un attributo diverso da quello di default.

Per quanto riguarda la struttura Pseudocode essa contiene la stringa da visualizzare per ogni linea dello pseudocodice (indicandone esplicitamente il numero) e la specifica grafica. Quest'ultima indica quali linee evidenziare, con quale colore, quale font utilizzare e in che punto visualizzare lo pseudocodice.

3.1.2 Specifica delle strutture dati

I file di input di ALViE sono anch'essi scritti in XML utilizzando una sintassi identica a quella delle strutture dati nei file di visualizzazione.

Ogni struttura dati possiede il proprio tag, che solitamente presenta almeno due attributi: `size`, che indica il numero di elementi della struttura dati, e `type`, che definisce il tipo di informazione presente nella struttura dati. All'interno del tag saranno poi presenti i tag che definiscono i singoli elementi. Anche questi tag contengono almeno due attributi: `id`, che rappresenta l'indice, o identificatore,

dell'elemento, e `value`, che ne indica il valore.

Alcune strutture, come i grafi, presentano attributi aggiuntivi, come l'orientazione, e possono anche distinguere tra elementi diversi, come nodi ed archi. Il seguente file XML può essere preso a esempio per la costruzione di un vettore di interi:

```
<array size="8" type="IntInformation">
  <element id="0" value="17"/>
  <element id="1" value="8"/>
  <element id="2" value="20"/>
  <element id="3" value="12"/>
  <element id="4" value="11"/>
  <element id="5" value="43"/>
</array>
```

Un discorso a parte va fatto riguardo agli alberi binari ed in particolare riguardo all'assegnazione degli indici. Nella specifica di un albero binario non sono presenti, in modo esplicito, i rapporti di parentela che intercorrono tra i nodi dell'albero. Le parentele (padre)→(figlio sinistro) e (padre)→(figlio destro) vengono infatti dedotte dall'indice di ogni nodo, che viene assegnato secondo lo schema in Figura 24.

Sia quindi x l'indice di un determinato nodo, il figlio sinistro di questo nodo

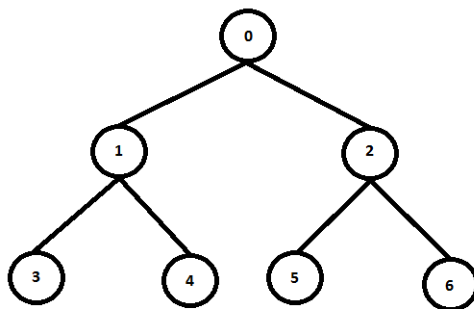


Figura 24: Assegnazione degli indici per i nodi di un albero binario nella specifica XML di ALViE.


avrà indice $2x + 1$, mentre il figlio destro $2(x + 1)$. Analogamente, se x è dispari, il padre di x avrà indice $\frac{x-1}{2}$, mentre se è pari $\frac{x}{2} - 1$.

3.2 ALViE4

ALViE4 rappresenta la quarta versione del sistema ALViE, nonché l'obiettivo principale di questo progetto. Le principali novità di questa versione sono l'esportazione delle visualizzazioni in pagine HTML5 e l'editor di strutture dati, che adesso andremo a vedere in maggior dettaglio.

3.2.1 HTML5

La nuova funzione di conversione delle visualizzazioni in pagine HTML5 rappresenta la principale novità introdotta nella quarta versione di ALViE.

Tramite il bottone  è possibile convertire una visualizzazione in pagina HTML5, che potrà poi essere inserita in un qualsiasi sito Internet.

Per esportare una visualizzazione in HTML5 è necessario innanzitutto generare una nuova visualizzazione o caricarne da file una precedente. A questo punto basterà cliccare sul bottone relativo all'HTML5 e selezionare il nome e la destinazione della nuova pagina HTML5 che conterrà la visualizzazione. Fatto ciò, il sistema si occuperà di creare la pagina HTML5 sulla base della visualizzazione indicata ed avviserà l'utente con un messaggio a video una volta che l'operazione sarà conclusa.

Per i dettagli implementativi riguardo all'intera procedura di esportazione in HTML5 della visualizzazione, si veda il Capitolo 4, che tratterà le tecniche utilizzate nel dettaglio.

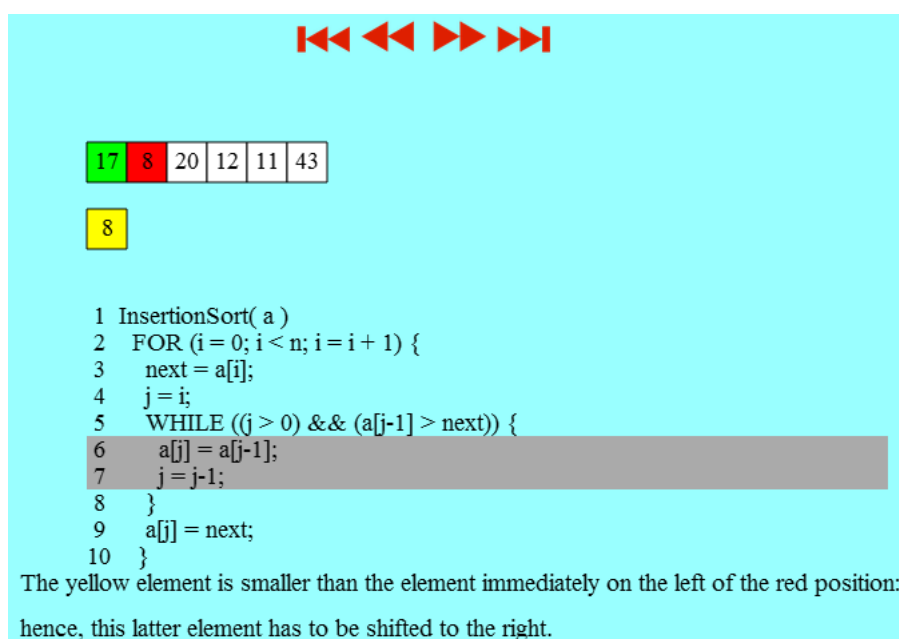


Figura 25: Una visualizzazione dell'algoritmo dell'InsertionSort esportata da ALViE in formato HTML.

3.2.2 Editor di strutture dati

Un'altra importante funzione aggiunta da ALViE4 è la possibilità di creare facilmente file di input tramite un editor di strutture dati. L'editor permette di creare i file XML di input, necessari per l'esecuzione e la visualizzazione degli algoritmi su ALViE, tramite un'interfaccia tabulare.

Le strutture dati disponibili nell'editor sono, attualmente, vettori, alberi binari, grafi (orientati e non), liste e matrici. I tipi di dato con cui è possibile inizializzare gli elementi di tali strutture dati sono invece valori booleani (vero o falso), caratteri, stringhe, numeri interi o numeri reali.

Per i vettori e le liste, verrà chiesta innanzitutto la dimensione della struttura dati, che dovrà quindi essere un intero positivo. Successivamente apparirà una tabella dove potrà essere inserito il valore per ogni elemento della struttura dati. La prima colonna di tale matrice è non modificabile e contiene gli indici relativi agli elementi della struttura dati, che vanno da 0 ad $n - 1$, con n la dimensione scelta.

Per gli alberi binari la situazione è molto simile, con l'unica differenza che la colonna degli indici risulterà vuota e modificabile. Si dovranno infatti inserire, oltre ai valori dei nodi dell'albero, anche gli indici relativi ad ogni nodo, seguendo la convenzione indicata in Sezione 3.1. Nel caso di indici non consistenti, verrà mostrato un messaggio di errore a video e si aprirà nuovamente la tabella di input per inserire nuovi valori.

Per quanto riguarda la matrice, bisogna specificare inizialmente sia il numero di colonne che il numero di righe. Verrà quindi visualizzata una matrice completamente vuota e modificabile della dimensione richiesta.

Infine per i grafi viene richiesto di scegliere se si vuole un grafo orientato o non orientato, quindi si passa all'inserimento del numero dei nodi e del numero di archi presenti tra di essi. Dopodiché verranno mostrate due tabelle in sequenza: la prima è relativa ai nodi del grafo e vi si dovrà indicare il valore di ogni nodo e le sue coordinate (X ed Y) espresse in pixel (vedi Figura 26); la seconda tabella serve invece per la definizione degli archi, indicando l'id del nodo di partenza, l'id del nodo di arrivo e l'etichetta dell'arco. Si osservi che, nel caso di grafi non orientati, il nodo di partenza e quello di arrivo sono interscambiabili. Inoltre la colonna delle etichette è opzionale: nel caso in cui non servano etichette di alcun tipo basterà lasciare la colonna in bianco.

La creazione di ogni struttura dati viene gestita da una classe Java diversa, detta *InputWriter*. Una volta scelta la struttura dati da creare, viene applicato il design pattern del *factory* per invocare l'editor corretto.

Inoltre sono presenti anche delle classi, dette *InputChecker*, per il controllo del tipo di input inserito. Se ad esempio si è scelto di creare un vettore di elementi interi e, nella tabella di creazione, viene inserito un valore non intero, come

una stringa, allora verrà mostrato a video un messaggio d'errore, permettendo all'utente di reinserire nuovi valori.

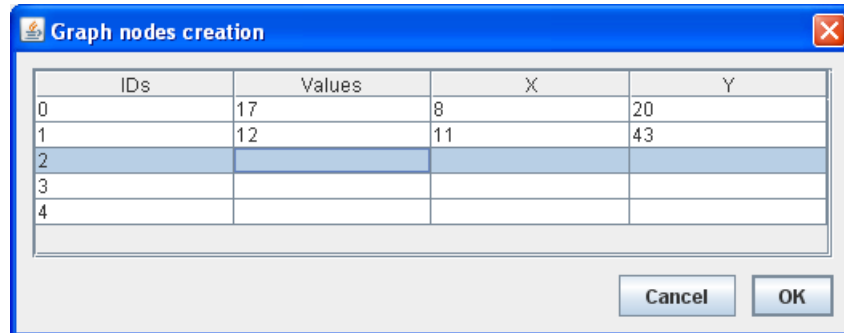


Figura 26: La creazione di un grafo orientato a valori interi tramite l'editor di strutture dati di ALViE4.

3.3 UN ESEMPIO PRATICO

Vedremo, in questa Sezione, un breve esempio su come costruire l'algoritmo del SelectionSort in ALViE e su come ottenerne una visualizzazione.

Innanzitutto si deve implementare l'algoritmo del SelectionSort in Java, utilizzando le strutture dati di ALViE e inserendo gli eventi interessanti (*interesting events*) all'interno del codice, come di seguito:

```

Array<I> a;
private int i, indiceMinimo, j, n;
private I minimo;
VisualSelectionSort vss;

private void selectionSort() {
    vss.init();
    vss.ieStart();
    for (i = 0; i < n; i++) {
        minimo = a.elementAt(i);
        indiceMinimo = i;
        vss.ieSelectionStarted(i);
        for (j = i + 1; j < n; j++) {
            if (a.elementAt(j).isLessThan(minimo)) {
                minimo = a.elementAt(j);
                indiceMinimo = j;
                vss.ieSelectionChanged(j);
            }
        }
        vss.ieSwapToBeDone();
        a.setAt(a.elementAt(i), indiceMinimo);
        a.setAt(minimo, i);
        vss.ieSwapDone(i);
    }
    vss.ieEnd();
}

```

Le funzioni che iniziano per *ie* sono, appunto, gli eventi interessanti di cui parlavamo prima. Ogni chiamata di queste funzioni definisce, quindi, un passo della visualizzazione. L'oggetto `VisualSelectionSort`, invece, è un'istanza della classe grafica del `SelectionSort`, che definisce e raccoglie le funzioni degli eventi interessanti.

Vediamo, a titolo d'esempio, la funzione `ieSelectionChanged()`, che viene invocata dall'algoritmo quando viene aggiornato il minimo corrente del vettore residuo, come mostrato in Figura 27:

```
private void ieSelectionChanged(int i) {
    if (tmp >= 0) {
        aColor[tmp] = getResource("aColor");
    }
    aColor[i] = getResource("currentMinimumColor");
    tmp = i;
    aDrawer.startStep(step++);
    aDrawer.draw(aIndex, aColor, getResource("ieSelectionChanged"));
    pseudocodeDrawerUtility.draw("", new int[] { 6, 7 });
    aDrawer.endStep();
}
```

Quello che viene fatto in questa funzione è impostare il colore del precedente minimo (`tmp`) al colore di default, mentre il colore del nuovo minimo viene impostato al colore scelto per il minimo nel file di configurazione. Quindi viene aumentato il numero di step e viene scritto il codice XML del nuovo vettore, tramite la funzione `draw()`. La stringa passata come terzo argomento rappresenta il messaggio da visualizzare durante quello step di visualizzazione, che viene indicato nel file di configurazione dell'algoritmo. Infine, prima della terminazione dello step, vengono scritti nel file XML le linee relative allo pseudocodice, evidenziandone gli elementi di indice 6 e 7, ovvero le linee 7 e 8 dello pseudocodice.


In linea di massima, le funzioni che definiscono gli eventi, devono aggiornare l'aspetto grafico delle strutture dati coinvolte ed indicare quali strutture dati visualizzare.

Una volta definito l'algoritmo c'è bisogno di definire il file `.properties` di configurazione. Questo file, specifico per ogni algoritmo, definisce dettagli grafici, come le dimensioni degli oggetti, i colori utilizzati o le font delle scritte. Non ci dilungheremo elencando tutti i campi del file di configurazione, ma di seguito si possono vedere alcuni dei campi più importanti:

- `pseudocodeFileName`: indica il nome del file XML contenente la specifica dello pseudocodice;
- `algorithmName`: indica il nome dell'algoritmo;
- `algorithmCategory`: indica il nome della categoria in cui inserire l'algoritmo nel menu di scelta del sistema;

- `algorithmFileName`: indica il nome del file .class contenente la specifica dell'algoritmo.

Definito l'algoritmo e costruito il file di configurazione, è finalmente possibile avviare il sistema e produrre una visualizzazione.

Avviato il sistema, si deve cliccare sull'icona  e quindi selezionare l'algoritmo appena costruito. Verrà allora chiesto di selezionare il file XML su cui scrivere la specifica della visualizzazione ed il file (o i file) XML di input, necessario all'esecuzione dell'algoritmo. Si osservi che nel caso in cui si voglia inserire un nuovo input, è necessario averlo costruito prima, manualmente o tramite l'editor di strutture dati. Una volta terminata l'esecuzione dell'algoritmo, un messaggio a video informerà l'utente e la visualizzazione verrà infine mostrata.

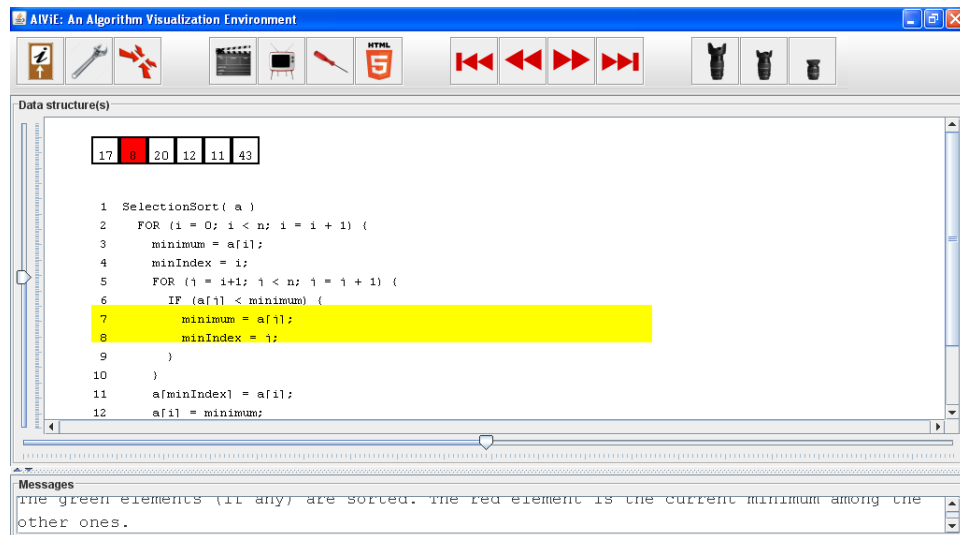


Figura 27: Uno step dell'esecuzione dell'algoritmo del SelectionSort all'interno del sistema AIViE.

3.4 SVILUPPI FUTURI

Sicuramente il supporto al Web e l'editor di strutture dati hanno fatto della quarta versione di AIViE un valido strumento per la visualizzazione di algoritmi. Queste novità aprono, però, anche un orizzonte di nuove possibilità per gli sviluppi futuri del sistema.

In futuro, ad esempio, si potrà eseguire AIViE direttamente sul Web, in modo da poter ottenere visualizzazioni su pagine HTML senza dover dipendere dalla versione Desktop del sistema. Una possibile implementazione potrebbe essere quella tramite CGI (*Common Gateway Interface*, ovvero *Interfaccia per Gateway*

Standardizzata): in una pagina Web iniziale saranno presenti una serie di menu ed opzioni per selezionare l'algoritmo da visualizzare ed indicare (o creare) l'input relativo. Dalle scelte fatte verrà generata una stringa di query contenente tutte le informazioni necessarie. Questa query sarà quindi inviata al programma CGI, che darà istruzioni al server per l'esecuzione dell'algoritmo e l'esportazione della visualizzazione in HTML5. Quindi, una volta generata la pagina della visualizzazione sul server, il programma CGI la invierà nuovamente alla macchina client, che potrà visualizzare l'esecuzione dell'algoritmo sul browser. Un altro possibile sviluppo, strettamente correlato a quello appena esposto, è la possibilità di inviare, sempre tramite Web, i propri algoritmi (non presenti, quindi, nella libreria di ALViE), che verranno eseguiti sul momento dal sistema sul server.

Altri possibili scenari futuri potrebbero prevedere l'utilizzo di librerie grafiche 3D per il supporto a visualizzazioni tridimensionali, oppure garantire una maggior elasticità degli algoritmi per quanto riguarda il linguaggio e la specifica della visualizzazione: il processo di produzione della visualizzazione potrebbe infatti essere implementato con un certo grado di automazione.

DA ALViE AD HTML5

Concludiamo questo studio analizzando nel dettaglio come avviene l'esportazione delle visualizzazioni di ALViE in HTML5.

Come abbiamo visto nel Capitolo 3, ALViE memorizza le visualizzazioni in appositi file XML, che definiscono ogni singolo passo di visualizzazione. Inoltre, sia nel caso in cui l'utente voglia generare una nuova visualizzazione, sia che voglia caricarne una vecchia, il sistema disporrà del file XML corrispondente. Quindi, osservando che la struttura di un file XML e di un file HTML sono molto simili tra loro, per implementare l'esportazione della visualizzazione in HTML5 si è deciso di definire un opportuno compilatore da XML ad HTML5. Questo compilatore XML-HTML5 avrà quindi il compito di tradurre le specifiche della visualizzazione nel file XML in comandi Canvas nel file HTML5.

4.1 SPECIFICA HTML5 DELLE VISUALIZZAZIONI

Analizziamo innanzitutto come viene strutturata la pagina di visualizzazione HTML5.

Innanzitutto troviamo, come in tutte le pagine HTML, il tag `<html>` che racchiude tutto il codice, e più precisamente i tag `<head>` e `<body>`. Il tag `<head>` contiene il tag `<script language="JavaScript">`, che definisce appunto uno script in JavaScript, suddiviso in una serie di funzioni. Il tag `<body>`, invece, contiene la definizione del Canvas e dei quattro bottoni di navigazione, indicandone l'origine dell'immagine utilizzata, le dimensioni e la funzione JavaScript associata al click su ognuno di essi.

Tramite l'attributo `onLoad="start();"`, del tag `<body>`, viene richiamata la funzione JavaScript `start()`, che permette l'inizializzazione della visualizzazione, disegnando sulla pagina il primo step.

Ad ogni passo dell'algoritmo corrisponde una funzione JavaScript, della forma

```
function stepX(),
```

dove X rappresenta il numero del passo di visualizzazione. La funzione relativa al primo passo sarà quindi `step0()`.

A questo punto, tramite i quattro bottoni di navigazione in cima alla pagina, è possibile spostare la visualizzazione su un altro step. Al click di ogni bottone è associata una funzione JavaScript che calcola la funzione `stepX()` da richiamare in

base al numero di step attuale e al tipo di funzione richiamata. Le funzioni di navigazione sono quindi quattro, `previous()`, `next()`, `first()` e `last()`, che richiamano, rispettivamente, il passo precedente, il successivo, il primo o l'ultimo della visualizzazione. Nel caso in cui la visualizzazione sia sul primo step e venga richiamata la funzione `previous()`, un messaggio d'errore verrà visualizzato dal browser; analogamente accadrà per l'ultimo step. Le quattro funzioni di navigazione si avvalgono della funzione `eval(str)` per richiamare la giusta funzione di step: questa funzione prende come input una stringa `str` e richiama la funzione corrispondente a tale stringa.

Ogni funzione `stepX()` dovrà quindi occuparsi, innanzitutto, di ripulire l'area del Canvas dal passo precedente, tramite la funzione `contesto.clearRect(0, 0, canvas.width, canvas.height)`. Dopo aver ripulito l'intero Canvas, verranno richiamate opportune primitive grafiche per produrre il disegno vero e proprio, comprensivo di strutture dati, pseudocodice e messaggio.

Viene anche definita una funzione `writeMessage(contesto, messaggio, X)`, con `X` punto di partenza sull'asse delle ascisse, per poter disegnare sul Canvas il messaggio. È necessaria una funzione a sé stante, in quanto i messaggi più lunghi devono essere spezzati in più linee, che è appunto ciò di cui si occupa la funzione. In caso contrario, messaggi troppo lunghi produrrebbero un'area del Canvas esageratamente lunga, rendendone scomoda la lettura.

4.2 COMPILATORE XML-HTML5

Il compilatore XML-HTML5 che è stato scritto per questo progetto traduce linea per linea il contenuto del file XML della visualizzazione in comandi Canvas da scrivere nella pagina HTML. Vengono quindi aperti due stream (o flussi) dal sistema: uno in lettura dal file XML ed uno in scrittura sul file HTML. L'intero processo di traduzione del file XML della visualizzazione è divisibile in due fasi: una fase di parsing ed una fase di compilazione.

4.2.1 Fase di parsing

La prima fase, quella di parsing del file XML, consiste nell'analizzare il file XML della visualizzazione ed estrarne le informazioni contenute, organizzandole in oggetti dalla struttura nota e facilmente accessibili. In questo senso, sono stati analizzati, per questo studio, due diversi approcci per il parsing dei file XML: utilizzando un parser XML generico oppure utilizzando un parser scritto ad-hoc per la traduzione XML-HTML.

Il parser XML generico utilizzato in questo studio è detto *Digester*, ed è sviluppa-

to e distribuito dalla Apache. Gli oggetti in cui archiviare le informazioni estratte dal file XML vengono invece detti *Bean*. Appositi Digester si occuperanno di ogni tag incontrato, incapsulando le informazioni estratte in un oggetto Bean relativo al tag designato per il parsing. Ogni Digester dovrà essere inizializzato con apposite regole che gli indichino come estrarre e strutturare gli attributi di ogni tag incontrato. I Bean, poi, conterranno le informazioni estratte, fornendo anche metodi accessori per ottenere i valori dei vari attributi in un secondo momento.

Gli oggetti Bean sono organizzati secondo una struttura gerarchica, rispecchiando l'organizzazione del file XML: ogni StepBean contiene, oltre ai suoi attributi, anche un insieme di StructureBean, relativi alle strutture contenute nel tag `<step>`; analogamente ogni StructureBean conterrà il Bean del messaggio ed i Bean della struttura dati e della struttura dati grafica (ad esempio, ArrayBean e VisualArrayBean), e così via. L'insieme di tutti gli StepBean definisce, quindi, il file XML in modo completo, con tutte le informazioni in esso contenute ben organizzate e facilmente accessibili.

Definendo, invece, un parser ad-hoc per l'esportazione di visualizzazioni ALViE in HTML, si ha che le informazioni ottenute dal parsing dei vari tag non vengono archiviate in oggetti di tipo Bean, ma direttamente in vettori e variabili primitive, pronte per essere utilizzate ai fini della compilazione. Inoltre, la fase di parsing può essere fusa con quella di compilazione, alternando le due per ogni tag incontrato. Per contro, scrivere e maneggiare un parser ad-hoc è sicuramente più complesso rispetto all'utilizzo di un parser XML generico, scritto e testato da terzi.

Comparando le due soluzioni su un calcolatore Netbook Packard Bell (CPU Intel Atom N280 @1.66 GHz, 1 GB di memoria RAM, Sistema Operativo Windows XP Home Edition SP3) si è potuto osservare che utilizzando il Digester come parser XML i tempi di compilazione raddoppiano rispetto alla soluzione ad-hoc. Questo è dovuto al fatto che la gestione del Digester stesso e degli oggetti Bean richiede sicuramente più risorse, rispetto alla gestione di qualche vettore. Inoltre, utilizzando il Digester è necessario dividere le fasi di parsing e di compilazione, eseguendo di fatto una compilazione in due passate (una per analizzare il file XML ed una per analizzare i Bean). Il parser ad-hoc viene invece fuso assieme al compilatore: ogni gestore di tag, come vedremo nella prossima Sezione, esegue il parsing del tag gestito ed utilizza subito le informazioni ottenute.

A titolo d'esempio, eseguendo dei test sulla macchina sopra citata, l'algoritmo LCS (Longest Common Sequence), con una stringa di 10 caratteri ed una di 7 come input, richiede 3.6 secondi di compilazione con il parser ad-hoc, mentre ben 8.5 con il Digester. Analogamente, l'algoritmo di Dijkstra, eseguito su un grafo con 8 nodi, passa da 1 secondo di compilazione a circa 2.3 con il Digester. Si è quindi deciso di mantenere il parser ad-hoc, che permette al sistema di

migliorare notevolmente le prestazioni in fase di compilazione XML-HTML5.

4.2.2 Fase di compilazione

Analizziamo quindi la compilazione vera e propria: ogni tag XML ha, all'interno del compilatore, una corrispondente classe adibita alla gestione di quest'ultimo. Queste classi sono dette *TagHandler*. Ogni volta che il compilatore si imbatte nell'apertura di un nuovo tag, viene invocata la classe *TagHandlerFactory* la quale, implementando il design pattern del *factory*, seleziona il giusto *TagHandler* per la gestione del tag.

La gestione dei tag avviene quindi, da parte del compilatore, su più livelli, o gerarchie: gestori di livello superiore (o esterni) corrispondono ai tag più esterni del codice XML, come ad esempio `<algorithm> O <step>`; i gestori di livello inferiore corrispondono invece a quei tag situati all'interno di molti altri tag e che solitamente non ne contengono nessun altro, come `<element> O <node>`.

Il gestore del tag `<step>`, ad esempio, si occupa di definire la funzione `stepX()` nel file HTML e inserire i comandi per la pulizia del Canvas. Quindi eseguirà un ciclo *while* in cui invoca il *TagHandlerFactory*, finché non si imbatte nella chiusura del tag `<step>`, ovvero `</step>`. Una volta usciti dal ciclo *while*, e quindi raggiunta la chiusura del tag, il gestore si occuperà di scrivere le informazioni necessarie alla stampa del messaggio di `step` sul file HTML.

I gestori dei tag di tipo `<element>`, invece, non effettuano alcuna scrittura sul file, in quanto devono prima essere raccolte informazioni su tutti gli elementi di una struttura dati per avere informazioni sufficienti a poter disegnare quest'ultima. Il gestore di questo tipo di tag, quindi, si limita a passare le informazioni, raccolte dal parser, al gestore di livello superiore, che sarà il gestore di un tag di struttura, come `<array> O <matrix>`.

Una volta ottenute le informazioni necessarie dai tag inferiori, il gestore del tag grafico della struttura (ad esempio `<visualArray>`) avrà finalmente tutte le informazioni necessarie per ricavare le primitive grafiche che produrranno il disegno nella pagina HTML.

Spesso però, le informazioni raccolte dai singoli elementi non definiscono in maniera esplicita le operazioni grafiche necessarie. Nella specifica XML di un vettore, ad esempio, vengono indicate le coordinate d'origine dell'intero vettore, non di ogni singolo elemento. Tutti questi dati, *impliciti* nella specifica XML, dovranno quindi essere calcolati, tramite apposite funzioni, per poter disegnare nella pagina i singoli elementi grafici che compongono il disegno finale.

4.2.3 *Pesantezza della pagina vs. pesantezza di compilazione*

Ha senso, a questo punto, domandarsi a chi spetta l'onere di calcolare tutti questi dati impliciti. Le scelte sono chiaramente due: il calcolo può essere fatto *a monte*, ovvero dal compilatore, oppure *a valle*, cioè dal browser HTML che visualizza la pagina.

Se viene scelto di effettuare questi calcoli lato browser, si avrà sicuramente una compilazione più veloce. Il compito del browser verrà, invece, appesantito, rendendosi necessaria l'occupazione di risorse per il calcolo di questi dati impliciti. Questo approccio permette, tuttavia, di riutilizzare il codice necessario alla visualizzazione di stessi oggetti grafici. Può, ad esempio, essere definita una funzione che disegni un quadrato, una che disegni un vettore o una che disegni un intero grafo. Se in una visualizzazione appaiono, ad esempio, più vettori, la visualizzazione di ogni nuovo vettore si tradurrà, nella pagina HTML, in un'unica invocazione della funzione adibita a disegnare vettori. Quindi questa scelta velocizza il compilatore, rallenta il browser e alleggerisce la pagina.

Scegliere, invece, di effettuare i calcoli *a monte* implica, chiaramente, un maggior costo computazionale del processo di compilazione. D'altro canto, la pagina HTML verrà alleggerita, per quanto riguarda il costo computazionale, in quanto il compilatore potrà scrivervi direttamente le primitive grafiche necessarie, senza bisogno di ulteriori calcoli. Tuttavia, utilizzare primitive grafiche, all'interno della pagina HTML, significa essere costretti a ripetere il codice necessario al disegno di uno stesso oggetto. Se la visualizzazione deve, ad esempio, disegnare più vettori, ognuno dei quali disegnato da (indicativamente) 10 primitive grafiche, si avranno nella pagina HTML 10 linee di codice per ogni nuovo vettore da visualizzare. Si vede che, rispetto al caso precedente, le dimensioni della pagina risultano aumentate di circa un fattore m , dove con m si indica il numero medio di primitive necessarie alla visualizzazione di un oggetto grafico elementare. Riassumendo: questa scelta ci porta ad una compilazione più lenta, una visualizzazione più veloce ed una pagina Web più pesante.

Non esiste, come si può vedere, una scelta ottimale: le due strade presentano sia pregi che difetti. Quale scelta intraprendere dipende allora dal tipo di applicazione per cui si è reso necessario il compilatore.

Nel nostro caso, ovvero la visualizzazione di algoritmi sul Web, si è ritenuto più efficiente optare per i calcoli dei dati *a monte*. Si è ipotizzato, infatti, che siano di più le volte in cui una visualizzazione viene eseguita (ovvero visualizzare i vari step che la compongono), rispetto alle volte in cui essa viene creata (tramite il compilatore) o caricata tramite Internet. Le dimensioni della pagina HTML sono infatti inversamente proporzionali al tempo necessario per il suo download.

CONCLUSIONI

Concludiamo questo studio con alcune considerazioni generali su quanto trattato.

La quarta versione di AlViE, nata sulla base delle nuove funzioni implementate nel corso di questo lavoro, risulta essere molto in avanti rispetto alla versione precedente: il supporto al Web permette la condivisione su larga scala delle visualizzazioni generate, facilmente inseribili in un qualsiasi sito Web. All'interno del libro di testo Strutture di Dati e Algoritmi (seconda edizione; Pierluigi Crescenzi, Giorgio Gambosi, Roberto Grossi, Gianluca Rossi; Pearson) sono già state utilizzate, a scopo didattico, diverse visualizzazioni prodotte con AlViE. Inoltre, sul sito Web di suddetto libro (http://wps.pearsoned.it/crescenzi_strutture-dati-algoritmi2/220/56566/14480998.cw/index.html) si possono trovare alcune visualizzazioni di AlViE esportate come pagina HTML, ai fini di supportare lo studio e la comprensione delle principali strutture ed algoritmi.

L'editor di strutture dati, invece, aggiunge un nuovo livello d'interazione utente-sistema, facilitando la specifica degli input. Con questa nuova versione vengono anche alleggerite le conoscenze richieste all'utente che voglia definire la propria visualizzazione: adesso, infatti, non è più richiesto in alcun modo che l'utente capisca e gestisca la specifica XML delle strutture dati, che saranno completamente occultate agli occhi di quest'ultimo.

AlViE si sta quindi avviando verso una maggior accessibilità e semplicità d'utilizzo, dimostrando di avere buone probabilità di inserirsi nel contesto mondiale dei sistemi di visualizzazione di algoritmi, tenendo testa ai maggiori esponenti del settore, come JHAvÉ o OpenDSA.

Concludendo, il lavoro svolto, dalla costruzione del compilatore e dell'editor di testo, allo studio dell'universo dei sistemi di visualizzazione di algoritmi, si è dimostrato essere molto interessante, a livello sia pedagogico che personale. Questo lavoro è riuscito, infatti, a conciliare pratica e teoria, nel campo della visualizzazione di algoritmi, e, più in generale, dell'Informatica.

BIBLIOGRAFIA

- [1] Survey of Algorithmic Animation Platforms. URL http://www.diss.fu-berlin.de/diss/servlets/MCRFileNodeServlet/FUDISS_derivate_000000001358/03_Kap_2.pdf. (Cited on pages 5, 6, 7, 9, and 11.)
- [2] URL <http://dev.opera.com/articles/view/svg-or-canvas-choosing-between-the-two/>. (Cited on page 37.)
- [3] Price Blaine A., Baecker Ronald M., and Small Ian S. A Principled Taxonomy of Software Visualization. *Visual Languages and Computing*, 4(3):211–266, Luglio 1994. (Cited on page 19.)
- [4] Ricardo Cabello (aka Mr. Doob). Voxels, Novembre 2010. URL http://mrdoob.com/129/Voxels_HTML5. (Cited on page 35.)
- [5] Hausner Alejo and Dobkin David P. GAWAIN: visualizing geometric algorithms with web-based animation. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, page 411–412, New York, NY, USA, 1998. ACM, ACM. ISBN 0-89791-973-4. doi:<http://doi.acm.org/10.1145/276884.276933>. (Cited on page 13.)
- [6] Yung Mei Ki Amy. On-line Animation for a Programming Language Course. Master's thesis, School of Computer Science and Software Engineering, Monash University, 2000. (Cited on pages 10, 11, and 15.)
- [7] Italiano G.F. Scarano V. Cattaneo G., Ferraro U. The Main Features of CATAI. URL <http://www.dia.unisa.it/cattaneo.dir/CATAI/node11.html>.
- [8] AT&T Tech Channel. L6 part I (Bell Laboratories' Low Level Linked List Language), Giugno 2012. URL <http://techchannel.att.com/play-video.cfm/2012/8/6/AT&T-Archives-L6-part-I-Bell-Laboratories-Low-Level-Linked-List-Language>. (Cited on page 3.)
- [9] Shaffer Cliff. OpenDSA: An Active-eBook for Data Structures and Algorithms. Sito di presentazione, Gennaio 2012. URL <http://algoviz.org/OpenDSA/>.
- [10] Guarini Gianluca. Da HTML 4 ad HTML5, Febbraio 2011. URL <http://www.html.it/pag/19263/da-html-4-ad-html5/>. (Cited on page 33.)
- [11] Roman Gruia-Catalin, Cox Kenneth C., Wilcox Donald, and Plun Jerome. *Pavane: A System for Declarative Visualization of Concurrent Computations*.

- Department of Computer Science, Washington University, Campus Box 1045, One Brookings Drive, Saint Louis, MO 63130-4899, Marzo 1991. (Cited on page 14.)
- [12] Stasko J.T. Tango: A Framework and System for Algorithm Visualization. *IEEE Computer*, 23:27–39, Settembre 1990. (Cited on page 3.)
 - [13] Stasko J.T. Animating Algorithms with XTANGO. *SIGACT News*, 23:67–71, Primavera 1992. (Cited on page 8.)
 - [14] Stasko J.T. Algorithm Visualization: Reflections and Future Directions. Technical report, Information Interfaces Research Group, School of Interactive Computing & Gvu Center, Georgia Tech, Settembre 2007.
 - [15] Naps Thomas L., Eagan James R., and Norton Laura L. JHAVÉ – An Environment to Actively Engage Students in Web-based Algorithm Visualizations. Lawrence University.
 - [16] Brown M.H. A System for Algorithm Animation. *Computer Graphics*, pages 177–186, Luglio 1984. (Cited on page 4.)
 - [17] Brown M.H. Exploring Algorithms Using Balsa-II. *IEEE Computer*, 21:14–36, Maggio 1988. (Cited on page 6.)
 - [18] Brown M.H. Zeus: A System for Algorithm Animation and Multi-View Editing. In *Visual Languages*, pages 4–9, New York, 1991. IEEE Workshop. (Cited on page 7.)
 - [19] Crescenzi P., Demetrescu C., Finocchi I., and Petreschi R. *LEONARDO: a software visualization system*. Dipartimento di Scienze dell’Informazione, Università degli Studi di Roma “La Sapienza”, Via Salaria 113, 00198 Roma, Italia. (Cited on page 12.)
 - [20] Baecker R. Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science. *MIT Press*, pages 396–381, 1998. (Cited on page 4.)
 - [21] van Kesteren Anne and Pieters Simon. *HTML5 differences from HTML4*. W3C, Ottobre 2012. URL <http://www.w3.org/TR/2012/WD-html5-diff-20121025/>.