Introduzione alle interfacce grafiche

Fabio Vitali

Introduzione

Oggi esaminiamo in breve:

- Come sono nate le interfacce grafiche
- Alcune teorie dell'apprendimento rilevanti
- Alcuni concetti base della programmazione di interfacce grafiche

Primi protagonisti

Doug Engelbart, con Augment (anni ‘60), dimostrò che il computer poteva essere uno strumento di produttività personale.

Seymour Papert, con il LOGO (anni ‘60), dimostrò che i computer potevano essere usati da non professionisti, addirittura da bambini.

Alan Kay, con FLEX (anni ‘60), e con Xerox Star (anni ‘70), dimostrò che la grafica poteva essere usata per le interfacce

Bill Atkinsons, realizzando il Macintosh Toolbox (primi anni ‘80), dimostrò che la grafica poteva essere realizzata in maniera efficiente con macchine “povere”.

Teorie dell'apprendimento

Alan Kay fu il primo ad ipotizzare l'uso sistematico della grafica. I primi studi risultarono però in sistemi sostanzialmente inutilizzabili.

Il LOGO gli fece capire che i meccanismi di apprendimento erano la chiave per organizzare l'interfaccia globale. Si concentrò su due pensatori in particolare:

- Jean Piaget (cognitivista svizzero, 1896-1980)
- Jerome Bruner ("Verso una teoria dell'istruzione", 1966)

La teoria piagetiana

I bambini non sono in grado di fare ragionamenti simbolici tradizionali fino all'età di 11 o 12 anni, ma sono invece molto abili in altri tipi di ragionamenti, anche avanzati, che coinvolgono nozioni di topologia, geometria differenziale, etc.

I bambini, crescendo dalla nascita alla maturità (adolescenza), passano attraverso stadi intellettuali diversi e successivi. Si possono ottenere cose molto complesse sfruttando la natura dei vari stadi, e causare problemi, frustrazioni ed ansie ignorandole.

Esempio dei due bicchieri d'acqua: i bambini sotto i dieci anni, anche vedendo versare dell'acqua da un bicchiere alto e magro ad un bicchiere basso e grosso, continueranno a pensare che nel bicchiere grosso ci sta più liquido.

I tre stadi piagetiani

- Lo stadio **cinestetico** è quello in cui il bambino impara a muoversi, a toccare, a spostare gli oggetti, ad afferrarli e a valutarne le caratteristiche strutturali e di robustezza
- Lo stadio **visuale** è quello in cui il bambino osserva l'aspetto esteriore degli oggetti, lo valuta, lo confronta, ne apprezza le caratteristiche visive più importanti (forma, colore, simmetria, etc.)
- Lo stadio **simbolico** è quello in cui il bambino valuta il significato, l'uso degli oggetti, si fa modelli mentali del mondo esterno e delle relazioni, e fa analisi simboliche non più sugli oggetti, ma su concetti astratti

L’elaborazione di Bruner (1)

Gli stadi piagetiani in realtà sono modalità sovrapposte e mai cancellate. I passaggi della crescita generano ***mentalità*** diverse, autonome ed indipendenti: ragionano in modo diverso, hanno abilità diverse, sono in contrasto le une con le altre

L’elaborazione dell’esperimento dei bicchieri d’acqua: se nascondo il bicchiere grosso, gli stessi bambini che vedendo insistono che ci sta più liquido capiranno che ci sta la stessa quantità. Scoprendo il bicchiere si ri-convincono del contrario.

E’ inoltre dimostrato che le mentalità bruneriane siano estremamente “modali”, e dopo aver preso il controllo lo lasciano con difficoltà: dopo aver risolto cinque problemi simbolici di fila, coloro che sono sottoposti a test vengono generalmente bloccati per ore a risolvere simbolicamente un problema che era banale risolvere visivamente.

L’elaborazione di Bruner (2)

Sebbene Bruner identifichi varie modalità e mentalità, le più rilevanti rimangono le mentalità create dai tre stadi piagetiani: realizzativa, iconica e simbolica.

- ***Mentalità realizzativa***: sapere dove ci si trova, in quale posizione, muoversi in un ambiente, manipolare oggetti
- ***Mentalità iconica***: riconoscere, confrontare, configurare, concretizzare
- ***Mentalità simbolica***: astrarre, concatenare catene di passi logici, dedurre

Le persone apprendono applicazioni della mentalità realizzativa con una parte del cervello sviluppata precedentemente alla parte che si occupa di applicazioni iconiche, e ancora prima di ragionamenti simbolici.

Lo Xerox Star

Doing with Images makes Symbols

- Doing - mouse - mentalità realizzativa - oggetti realizzati come oggetti manipolabili fisicamente
- Images - icone, finestre - mentalità iconica - oggetti che si differenziano e rassomigliano visivamente, confrontabili, comparabili.
- Symbols - SmallTalk - mentalità simbolica - oggetti che si prestano ad astrazioni, ragionamenti, modifiche e personalizzazioni.

Caratteristiche dello Xerox Star

Finestre sovrapposte

- permettono il confronto, facilitano la complessità fornendo contesti autonomi

Modelessness

- Si passa da una modalità all'altra senza atti di terminazioni speciali, semplicemente facendo click sulla finestra giusta

Object-Oriented

- l'oggetto fornisce informazioni sul tipo di azioni che è in grado di fare. Nasce la sintassi "selezione-comando"

Text editing

- come sbarazzarsi di modalità "insert" e modalità "replace"? Introducendo il concetto di selezione.

Alan Kay e le metafore

- Es: lo schermo come carta su cui scrivere, disegnare, cancellare

Suggerisce comandi, proprietà, azioni, interazioni

Ma non suggerisce la magicità dell'azione: vogliamo che la carta su computer sia altrettanto difficile da modificare e cancellare che la carta vera?

E' la magia - magia comprensibile - che conta, non l'adeguamento ad una realtà esterna

Meglio "illusione dell'utente" che "metafora"

- Es. il desktop come metafora deve sparire: le vere scrivanie si riempiono di carta, non si auto-ordinano, non danno aiuti quando si tratta di cercare qualcosa.

Alan Kay e l’end-user programming

Il LOGO dimostrava che anche bambini o utenti poco esperti potevano programmare i computer.

SmallTalk doveva fornire il terzo stadio dell’uso di un computer, la programmazione, anche a chi non fosse un programmatore professionista.

Il problema è che programmare è complesso, e richiede focalizzazione semantica che è difficile realizzare in via iconica e immediata

Alan Kay e gli agenti

Piccoli elementi di software che fanno cose in modo autonomo

Programmare allora sarebbe istruire gli agenti

Usano intelligenza, pazienza, ricerca esaustiva

Svolgono per noi compiti che richiederebbero attenzione ed intelligenza

La programmazione grafica

- Elementi di computer graphic
- Elementi di gestione degli eventi
- Elementi di controllo dell'interazione
- Tipologia dei widget
- Clipboard ed altre funzionalità dell'interfaccia

Elementi di computer graphic (1)

Modelli di immagine:

- Modello a tratti (stroke model): ogni oggetto del disegno è rappresentato come un tratto (oggetto grafico) con forma, colore, coordinate, spessore.
- Modello a pixel: ogni punto indirizzabile ha associato un colore.
- Modello a regioni: oggetti al tratto vengono usati per delineare delle regioni che possono essere riempite con colori costanti o blending.

Sistemi di coordinate

- Coordinate del device (pixel)
- Coordinate fisiche (cm, pollici, pica, font size)
- Coordinate del modello (metri, km, miglia)

Elementi di computer graphic (2)

Il canvas

- Più o meno tutti i modelli di disegno forniscono l'astrazione del canvas, superficie disegnabile
- Un canvas ha dimensioni, profondità, unità di misura.
- Ogni canvas fornisce metodi per disegnare forme e testo.

Forme

- Paths: linee, archi, spline, percorsi a segmenti
- Aree chiuse: rettangoli, cerchi, ellissi, altre forme chiuse
- Testo: nome del font, dimensione, stile, altre informazioni sul testo: baseline, leading, height, width, ascent, descent.

Clipping

- Clipping su aree regolari: il problema della chiusura (l'intersezione di rettangoli è un rettangolo, l'unione no!)
- Clipping su regioni irregolari

Elementi di gestione degli eventi (1)

Windowing system

- Nella maggior parte dei sistemi grafici, esistono più processi o thread che condividono lo schermo. Per mettere ordine a questa condivisione, esistono i sistemi a finestra.
- I sistemi a finestre divide lo schermo in rettangoli autonomi sovrapponibili (detti finestre) che contengono oggetti interattivi organizzati secondo una gerarchia esplicita.
- Il windowing system è poi responsabile di arbitrare l'input e lo spazio-schermo condiviso tra tutte le finestre presenti.

Elementi di gestione degli eventi (2)

Eventi

- L'input dell'utente, nonché molti tipi di occorrenze vengono definiti "eventi". L'evento viene ricevuto dal sistema operativo e passato all'applicazione come procedure call o, in casi più complessi, come messaggio inter-processo.
- Ogni evento è definito da un tipo, nonché varie ulteriori informazioni (posizione del mouse, tasto premuto, stato dei modificatori).
- Tra i tipi di eventi:
  - Mouse button (con o senza modificatori), double click, function buttons, mouse movement, mouse-enter e mouse-exit, keyboard, window events
- Il problema del redrawing: off-screen drawing, backing store (X), redraw events.

Elementi di gestione degli eventi (3)

Il main event loop

- I sistemi ad interfaccia grafica hanno dato inizio ad una nuova filosofia di programmi interattivi, in cui il controllo dello sviluppo dell'applicazione è lasciato all'utente.
- Il programma dunque, è controllato dagli eventi utente, ed ha un main sempre uguale, detto ***Main Event Loop***:

```
Main() {
    Initialize()
    while (! Time_to_quit) {
       Get_next_event(E)
       Dispatch_event(E)
    }
}
```

- Ogni sistema ha una coda di eventi per processo in cui vengono messi tutti gli eventi importanti per l'applicazione. Il main event loop estrae in ordine gli eventi e li processa.

Elementi di gestione degli eventi (4)

Filtrare gli eventi

- Spesso gli eventi gestiti dal sistema operativo sono di livello troppo basso per essere utili. Es.: eventi per la lettera ‘T’:
  - Key down 16 Tasto Control
  - Key up 16 Oops: non era giusto!
  - Key down 75 Tasto Shift
  - Key down 42 Tasto ‘t’
  - Key up 42 Tasto ‘t’
  - Key up 75 Tasto Shift
- Ovviamente è necessario filtrare gli eventi non interessanti, tenendo soltanto quelli che servono. Il main event loop diventa:

```
Main() {
    Initialize()
    while (! Time_to_quit) {
       Get_next_event(E)
       if (! Filtered_event(E){
         Dispatch_event(E)
       }
    }
}
```


Elementi di gestione degli eventi (5)

Nascondere il main event loop

- ***Approccio object-oriented***: esiste una classe application che si occupa di gestire il main event loop e le procedure di quit. Il main risulta allora:

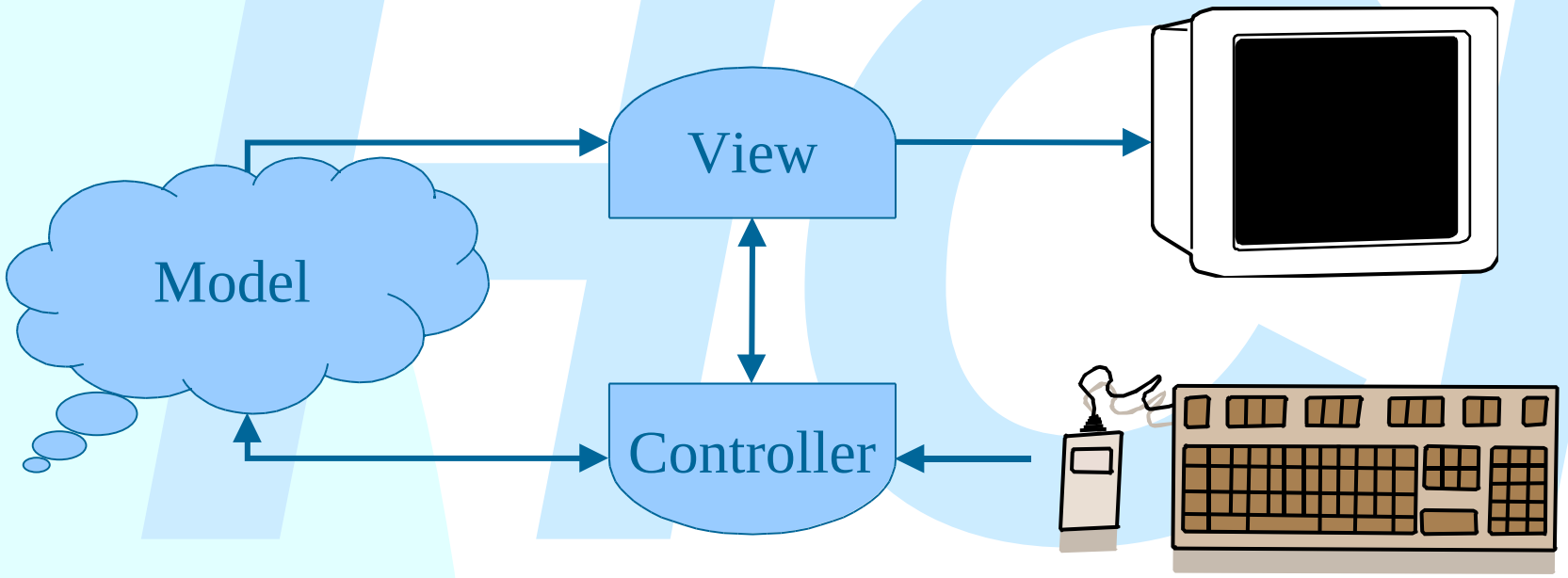
```
Main() {
   Application myApp = new Application() ;
   myApp.init() ;
   myApp.run();
}
```

- ***Approccio callback***: nel main si registra ogni tipo di evento ad ogni widget con l'indirizzo di una procedura da chiamare:

```
Main() {
   XtAddCallback(MyWidget,MyName,MyProcPtr,MyData);
   …
}
```


Controllare l’interazione (1)

L’architettura Model-View-Controller



Controllare l’interazione (2)

L’architettura Model-View-Controller

- Il **modello** è l’informazione che viene manipolata dall’applicazione, o anche solo dal widget.
- La **view** è la visualizzazione del modello sullo schermo. Possono esistere view molteplici per ogni modello.
- Il **controller** riceve gli input dall’utente e decide cosa significhino e cosa si debba fare.
- Poiché controller e view sono spesso molto vicini, vengono realizzati adesso con un’unica classe, con metodi come Paint() per gestire la view, e metodi come MouseUp() per gestire il controller.
- Vanno quindi implementate separatamente le classi che implementano il modello funzionale dell’oggetto, da quelle che implementano la view. Il controller può essere associato in molti casi alla view.

Controllare l’interazione (3)

L’architettura Model-View-Controller

- Perché non unire le tre parti in un’unica classe? Perché posso avere view diverse dello stesso modello (ad esempio, visione numerica e slider di una stessa variabile). Poi perché differenti architetture impongono view diverse allo stesso modello.

Aggiornamento del display

- Se modifico i valori del modello, c’è il problema di aggiornare adeguatamente la view. Il modo più semplice è ridisegnarla tutta, da sotto a sopra. Ma questo può essere molto complesso in certe situazioni.
- Un metodo alternativo, fornito da tutti i toolkit, è il meccanismo damage/redraw (ha anche altri nomi): la view specifica quali parti vanno ridisegnate, il windowing system identifica quali regioni sono effettivamente visibili, e chiede al modello di ridisegnare solo quelle regioni che vanno modificate.

Tipologia dei widget (1)

Si possono raggruppare i widget più comuni sulla base del loro modello:

- ***Pulsanti***: un valore discreto di controllo dell'attivazione di un comando. Non ha uno stato. Esistono molte view diverse anche per lo stesso modello: icone, pulsanti arrotondati, ecc.
- ***Check-box e radio button***: un singolo valore discreto, o booleano (check-box), o numerico e condiviso da più widget (radio button).
- ***Slider***: un valore in un intervallo continuo e limitato. Il modello contiene quantomeno gli estremi dell'intervallo e il valore attuale. A volte non esprime un valore, ma un intervallo; in questo caso alcuni toolkit permettono di visualizzare il sotto-intervallo in dimensioni proporzionate. Anche qui esistono molte view diverse.

Tipologia dei widget (2)

- ***Menu***: un valore tra una lista anche molto grande di valori possibili. Il valore può attivare un comando, o modificare una variabile. Ogni toolkit fornisce meccanismi per gestire elenchi molto lunghi, anche non contenibili in un'unica schermata: liste scrollabili e menu gerarchici sono un esempio. Anche qui esistono molte view diverse per lo stesso modello.
- ***Box di testo***: il modello è una stringa di testo. Ogni toolkit fornisce meccanismi standard di interazione: movimento, selezione, clipboard. La box di testo con modello numerico o data è un widget diverso? Per il momento no, ogni toolkit fornisce un'unica box di testo e sta ad un post-processing dell'applicazione rifiutare valori mal formati.

Tipologia dei widget (3)

Device astratti

- Lo studio dei widget di un'interfaccia grafica deriva da studi precedenti di ergonomia per una quantità di controlli usati nelle macchine fisiche: manopole, pulsanti, slider, tavolette varie, ecc.
- Alcuni sistemi usavano questi controlli fisicamente. Ci si rese conto che molti di questi device avevano caratteristiche comuni e potevano essere gestiti dallo stesso codice.
- Inoltre, in assenza di alcuni controlli fisici fondamentali, si pensò di simularne l'esistenza su schermo.

Tipologia dei widget (4)

Device astratti

- Da qui nacque il concetto di device astratto:
  - Device fisico: la tastiera, il mouse, e ogni altro tipo di oggetto fisico per l'I/O effettivamente presente sulla macchina
  - Device virtuale: la simulazione su schermo, o la reimplementazione con altro device fisico, delle caratteristiche di un device astratto.
  - Device logico: la descrizione del modello di un input. Permette di separare l'implementazione delle funzionalità del modello da quelle del controllo del device.
- I device astratti servono per rendere l'implementazione del software indipendente dalla gestione dei suoi meccanismi di input.
- Ad esempio, il comando di quit può essere visto come un unico device logico, a cui corrispondono vari device virtuali o fisici: una voce di menu, un tasto funzione, ecc.

Clipboard e altre funzionalità (1)

La clipboard

- Una grande invenzione del Macintosh è stato un meccanismo obbligato e centralizzato per lo scambio di informazioni tra applicazioni, la clipboard.
- Questo permette alle applicazioni di non dover implementare tutte le funzionalità desiderabili, né studiare meccanismi di scambio di informazione specifici, applicazione per applicazione.
- Le prime implementazioni della clipboard erano molto semplici: pochi formati erano gestiti, e il dato era passivo, e non si aggiornava automaticamente. Meccanismi successivi hanno notevolmente complicato e sofisticato questi meccanismi.
- La clipboard semplice è semplicemente una locazione in memoria, dove un blocco di informazione viene tenuto per essere incluso in un'altra applicazione. Al blocco è associato un tipo, che permetta all'applicazione ricevente di sapere se può o non può includere questo tipo di dati.

Clipboard e altre funzionalità (2)

La clipboard

- Una prima sofisticazione è quella di permettere all'applicazione di mettere più blocchi di tipi diversi, per permettere all'applicazione ricevente di scegliere il formato più consono alle proprie capacità ed esigenze. Ad esempio, uno spreadsheet metterà un formato proprio, un formato testo ed un formato immagine degli stessi dati.
- Demand-based pasting: a volte non conviene mettere molti formati diversi nella clipboard (ad esempio, in X ci possono essere molti scambi di informazione tra macchine diverse). Per questo motivo si usano meccanismi di paste in cui l'informazione da incollare non viene mai copiata finché non si sa in quale forma verrà richiesta dall'applicazione ricevente.
- Publish&subscribe (o OLE): un architettura per fare paste di informazioni vive ed aggiornabili tra un'applicazione e l'altra. Gli oggetti mantengono un contatto diretto con l'applicazione generante, e possono essere aggiornati ogi volta che l'utente lo richiede.

Clipboard e altre funzionalità (3)

Undo e redo

- I meccanismi di undo e redo sono importantissimi per permettere forgiveness e sperimentabilità di un'interfaccia. Per implementare meccanismi di undo e redo è necessario monitorare ogni comando disponibile nell'interfaccia.
- I comandi sono una chiusura logica di una serie di eventi da input: per disegnare una riga, ad esempio, ci saranno gli eventi **`mousedown, mousedrag, mouseup.`** Questi eventi corrispondono ad un singolo comando atomico.
- Un meccanismo di undo è basato su una history list, una lista di comandi atomici eseguiti in passato. Ogni undo è l'estrazione dell'ultimo comando presente in lista, ogni redo è il suo riposizionamento in lista.
- Un modello semplice di undo prevede, per ogni estrazione, di rieseguire tutti i comandi dal primo all'ultimo presenti nella lista. Questo è ovviamente lentissimo.

Clipboard e altre funzionalità (4)

Undo e redo

- Un secondo modello prevede che ad ogni comando sia possibile associare un comando inverso: ad ogni insert corrisponde un delete, ad ogni delete un insert, ad ogni shrink un expand, ecc. In casi particolari, può valere la pena di memorizzare una copia dell'intero blocco di dati prima della memorizzazione.
- Esistono due tipi di comandi non inseriti nella lista della history:
  - Comandi transienti: ad esempio, operazioni sulla finestra, scrolling, cambio di selezione, ecc., che non hanno effetto sui dati
  - Comandi irreversibili: stampa, salvataggio, ecc. hanno effetti non o difficilmente reversibili, e quindi non entrano nella storia dei comandi.
- Undo selettivo: se si ha la sicurezza che i comandi successivi non ne sono influenzati, può essere possibile rimuovere gli effetti di un'azione precedente all'ultima. Questo viene detto undo selettivo

Clipboard e altre funzionalità (5)

Undo e redo

- Redo intelligente: Nel modello semplice di undo, solo un comando tolto dalla lista può essere rimesso dentro con il comando undo. Se però i comandi sono espressi in maniera parametrica, è possibile aggiungere alla lista di history un nuovo comando mai eseguito dall'utente corrispondente all'ultimo comando eseguito su una nuova selezione.

Macro

- Una gestione sofisticata ed intelligente dei comandi, implementati in maniera parametrica, permette di realizzare con poco sforzo un linguaggio di macro, che corrisponde all'esecuzione controllata di una sequenza di comandi atomici.

Conclusioni

Oggi abbiamo parlato di

- Alcuni importanti personaggi nello sviluppo di interfacce grafiche
- La teoria piagetiana, quella bruneriana, e il loro sfruttamento per la realizzazione dello Xerox Star
- La programmazione basata su eventi
- Creazione, tipologia ed uso di widget ed altri aspetti dell'interfaccia grafica.

Riferimenti

- ***A. Kay, "User Interface: a Personal View", in B. Laurel, The Art of Human-Computer Interface Design, Addison Wesley, 1990***
- D. Olsen, Jr., ***Developing User Interfaces***, Morgan e Kaufman, 1998