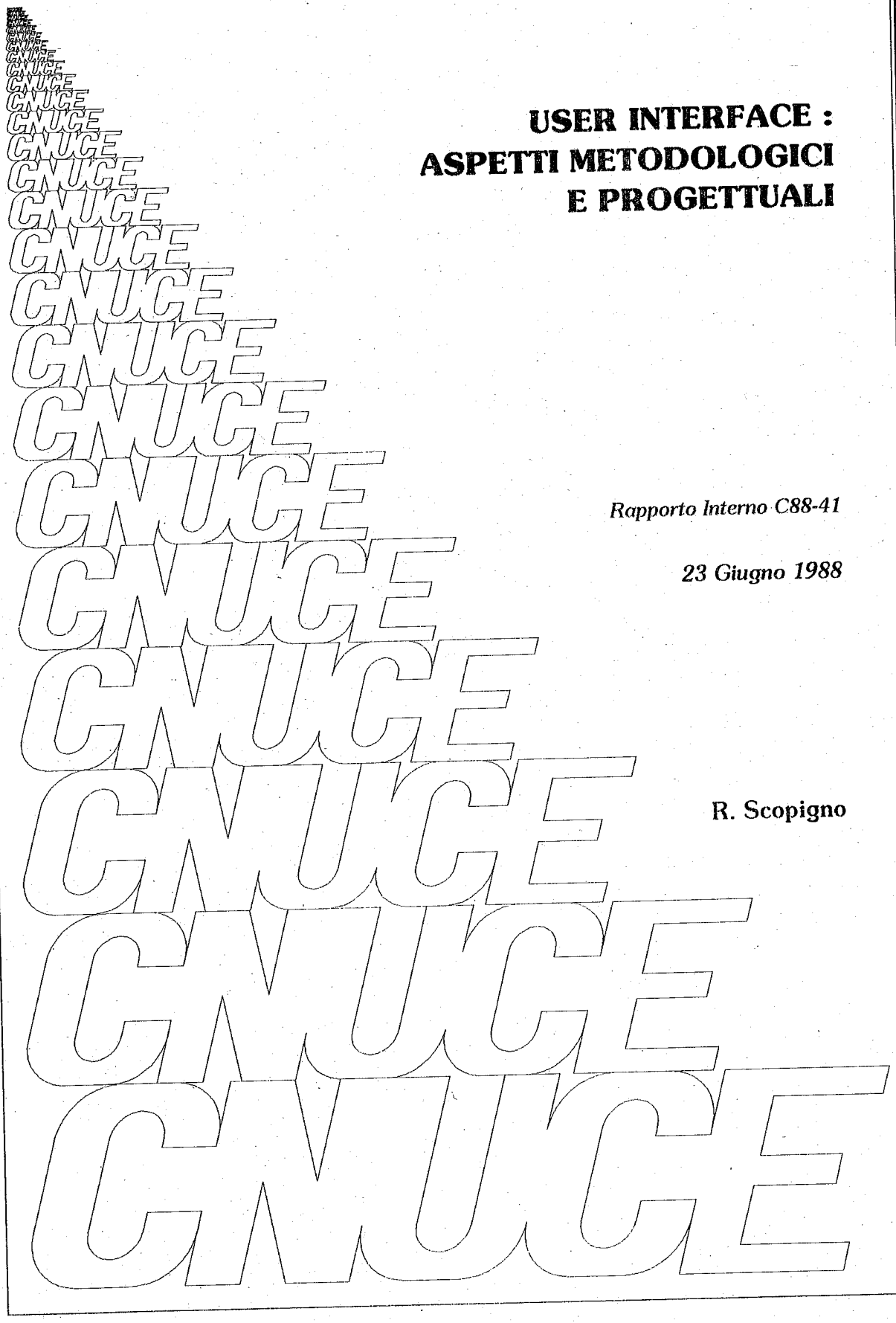

USER INTERFACE : ASPETTI METODOLOGICI E PROGETTUALI

Rapporto Interno C88-41

23 Giugno 1988

R. Scopigno

User Interface:
aspetti metodologici e progettuali.

R. Scopigno

Rapporto Interno C88-41

Materiale relativo al Seminario tenutosi presso
l'Istituto CNUCE il 23 Giugno 1988.

Istituto CNUCE
Consiglio Nazionale delle Ricerche

Via S. Maria 36 Pisa (ITALY)

User Interface :
aspetti metodologici e progettuali

R. Scopigno

"... The ideal situation, of course, would be to interact with the computer as if it were a helpful human being, perhaps chattering in a natural language. The computer interface may eventually metamorphose into a total sensory environments..."

A. Van Dam, 1984

Indice:

[1] Introduzione

I piu' recenti sviluppi delle tecnologie informatiche, soprattutto nel campo del personal computing, hanno evidenziato come la produttivita' nell'uso di sistemi informatici sia influenzata in modo determinante dalle caratteristiche dell'interfaccia uomo-macchina (Human-Computer Interface, HCI), in termini sia di facilita' d'apprendimento che di semplicita' d'uso del sistema [1].

Progettare un'interfaccia significa definire le modalita' di interazione che regolano la collaborazione tra le due entita' in gioco, l'uomo ed il computer, e di conseguenza in che misura ciascuna delle due debba avvicinarsi al modo d'essere dell'altra, imitandone comportamento e modalita' di lavoro. Date le caratteristiche peculiari della macchina "uomo" (abilita' nel monitoraggio, nel controllo, nel riconoscimento di forme; lentezza nell'elaborazione di informazione numerica e piu' o meno accentuata labilita' di memoria) e quelle della macchina computer (ottimale nella gestione ed elaborazione di grandi quantita' di informazioni e nella generazione di sintesi di tali attivita'), e' evidente come lo sforzo di "interfacciamento" debbe essere effettuato dal computer piuttosto che dall'utente umano. Cio' richiedera' la disponibilita' sia di stazioni di lavoro dotate di buone capacita' di elaborazione e di sofisticate potenzialita' grafiche, sia di software in grado di sfruttarle appieno e di supportare efficientemente l'interazione tra utente e sistema.

Un'interfaccia gestita da computer avra' buone possibilita' di essere considerata realmente "user friendly" solo se non stravolgera' il modello comportamentale dell'utente tipo, modello comportamentale dipendente generalmente sia dalle caratteristiche psicologico-comportamentali che dalle precedenti esperienze di uso di applicazioni similari.

La progettazione di una interfaccia non puo' quindi prescindere da una attenta analisi del comportamento dell'utente-tipo, svolta sia a livello di studio delle funzionalita' richieste da quest'ultimo all'applicazione che a livello di psicologia comportamentale. Lo sviluppo di user interface, e, in particolare, degli strumenti d'ausilio nello sviluppo di interfacce, richiede quindi molteplici studi e competenze (computer graphics, sistemi operativi, psicologia comportamentale, modalita' di reazione dell'uomo a colore, forme, movimento, etc.).

L'analisi scientifica del processo di progettazione di una user interface costituisce un'area di ricerca interdisciplinare che solo recentemente ha cominciato ad essere indagata, anche se con sempre crescente interesse. Risultato di tale attivita' e' stata la definizione di una serie di principi e di strumenti che guidano e facilitano il lavoro di sviluppo. In particolare, e' universalmente accettata la necessita' di una rigorosa separazione tra le funzionalita' dell'applicazione e quelle dell'associata user interface, consentendo lo sviluppo e l'evoluzione indipendente anche se correlata di entrambe le componenti.

[2] STILI E TECNICHE DI INTERAZIONE

> *"A subtle thing happens when everything is visible:* ***the display becomes a reality.*** *The user model becomes identical with what is on the screen. Objects can be understood purely in terms of their visible characteristics. Actions can be understood in terms of their effects on the screen. This lets users conduct experiments to test, verify and expand their understanding - the essence of experimental science."*
>
> *Smith et al., "Designing the Star User Interface", 1982*

Sono qui elencati una serie di principi e di tecniche che caratterizzano le interfacce "user friendly" (il termine tecnico indica la confortevolezza e la semplicita' d'uso dell'interfaccia) e che sono ritenuti validi in ragione sia della loro larga diffusione che della loro ampia accettazione nell'ambiente.

Pointing come alternativa al typing

L'ampio uso di icone o simboli grafici che, visualizzati sullo schermo, rappresentino gli oggetti o le operazioni gestite dal sistema e' una caratteristica comune a tutte le moderne interfacce grafiche ed ha originato una profonda differenziazione nelle modalita' di gestire il colloqio tra utente e computer.

Con il termine "**typing**" si intende l'approccio classico dei sistemi dotati di interfacce guidate dai comandi, sistemi in cui qualsiasi attivita' deve essere comandata digitando sulla tastiera un corrispondente codice (generalmente alfa-numerico). Tale approccio presenta due evidenti punti deboli: la scomodita' dell'uso della tastiera e soprattutto la necessita' di ricordare a memoria l'insieme di codici e sintassi associati ai comandi, il numero dei quali generalmente cresce al crescere della complessita' del sistema.

Una moderna interfaccia grafica supera tali difficolta' permettendo all'utente di selezionare comandi **puntando** stringhe presenti sullo schermo (i codici dei comandi che altrimenti avrebbero dovuto digitare), o anche puntando simboli grafici o icone che li rappresentano in forma spesso molto piu' facilmente intellegibile. Inoltre, in interfacce sofisticate, movimenti effettuati con il mouse possono comandare azioni mimando il normale modo di agire dell'utente nell'effettuare operazioni simili nel mondo reale. Si dira' in questo caso che il sistema utilizza delle **metafore** per la realizzazione di particolari comandi. L'utilizzazione di metafore permette in genere all'utente di comprendere il significato delle azioni in modo molto piu' immediato. Un esempio valido di cosa si

intende per metafora e' l'uso del "cestino" nei personal computer Macintosh della Apple. Il cestino del Macintosh e' la piccola icona raffigurante un bidone della spazzatura visualizzata nell'angolo in basso a sinistra dello schermo della macchina (fig.1). Per cancellare un file, operazione classica che a livello di un qualsiasi sistema operativo si descrive in (1) elimina il riferimento al file dalla directory (2) cancella fisicamente i dati, si utilizza nel Macintosh una metafora collegata alla ben nota funzionalita' del cestino; l'utente Macintosh vede quindi la cancellazione del file trasformarsi in (1) butta l'icona che rappresenta il file nel cestino (2) svuota il cestino. E' immediato notare l'intuitivita' dell'operazione di cancellazione risolta in questo modo.

Metafore ben studiate permettono inoltre che i modelli concettuali che gli utenti si formano nell'utilizzare l'interfaccia siano molto vicini al modello proprio dell'interfaccia stessa. Quest'ultima caratteristica e' fondamentale per l'efficacia dell'interfaccia: se l'utente e' portato in modo naturale a ragionare con le stesse modalita' utilizzate dal sistema, supera automaticamente la necessita' di tradurre dal suo modello a quello del sistema e l'interazione diviene ottimale.

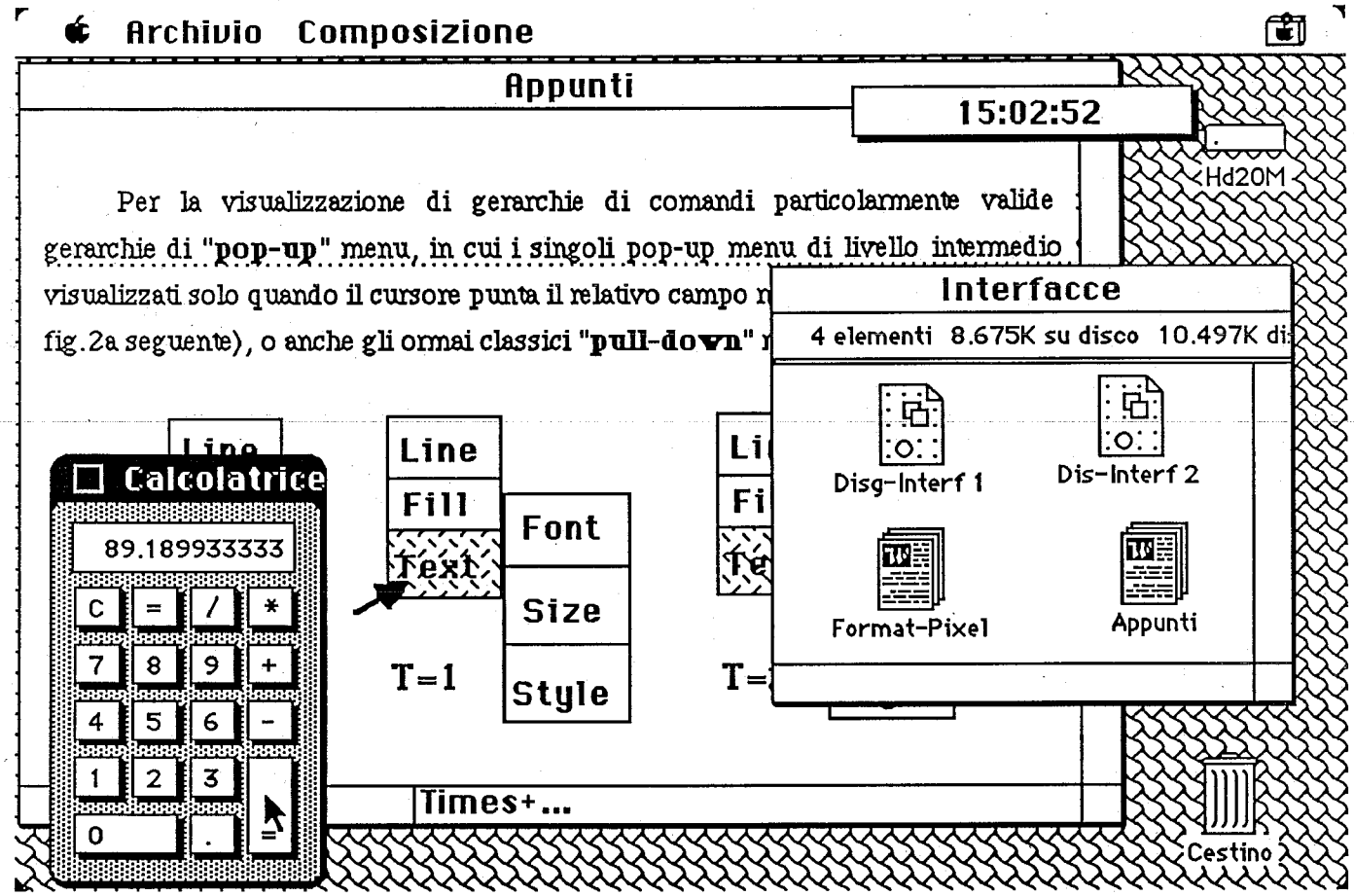

Fig.1 : Simulazione del tavolo di lavoro in ambiente multi-window.

WYSIWYG

Con questo criptico termine, acronimo di **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et, si indica la situazione in cui cio' che l'utente vede sullo schermo e' una diretta e veridica rappresentazione dello stato del sistema o dell'applicazione. Non rispettano questo assioma, ad es., i sistemi di composizione di testi o alcuni sistemi grafici che richiedono espliciti comandi di formattamento o di ridisegno per visualizzare gli effetti di comandi che abbiano causato modifiche nella paginazione del testo o nelle immagini visualizzate.

E' importante che sia le applicazioni che le interfacce rispettino l'approccio WYSIWYG poiche' se cio' che e' sullo schermo non riflette precisamente lo stato del sistema spesso l'utente puo' non comprendere chiaramente l'effetto dei comandi che utilizza, puo' non rendersi conto del modo in cui i commandi sono effettivamente interpretati dal sistema o puo' non essere in grado di effettuare decisioni valide riguardo alle successive azioni da intraprendere.

Economia nel numero di comandi

Lo stesso tipo di azione, ad esempio una doppia pressione (click) sul bottone del mouse che punta una icona, puo' essere associato ad una serie di operazioni che sono semanticamente simili nonostante operino su oggetti diversi. Un esempio e' l'uso del doppio click utilizzato per comandare sia l'apertura di un file (nel caso che il mouse punti l'icona relativa ad un file) che per richiedere la lista dei file in una sotto-directory (nel caso che il mouse punti l'icona di una directory). Tale approccio e' del tutto naturale per l'utente, che in entrambi i casi vuole vedere cosa e' contenuto nell'oggetto puntato. In tale modo lo stesso metodo di interazione diviene il modo in cui l'utente puo' indicare tutta una classe di operazioni simili, eliminando la necessita' di una moltitudine di comandi con codice diverso pur se con significato analogo.

Il concetto del doppio click risponde ottimamente al principio dell'economia del numero di comandi, che richiede, ove possibile, la realizzazione di comandi molto generali applicabili a oggetti diversi e di ristretti insiemi di comandi specializzati utilizzabili solo su tipi particolari di oggetti.

I comandi possono essere classificati in due categorie: **pre-fix** o orientati all'azione(l'utente seleziona prima l'operazione e quindi l'oggetto a cui tale operazione va applicata) e **post-fix** o orientati agli oggetti (l'utente seleziona prima un oggetto e quindi sceglie l'operazione da applicargli).

Vantaggi evidenti dell'approccio post-fix, scelto sempre piu' spesso nella definizione di interfacce, sono i seguenti :

(1) l'interazione e' piu' semplice; generalmente si puntano una serie di oggetti e quindi si invoca un comando, non e' necessario ricordare l'ordine sintattico dei parametri;
(2) l'utente puo', anche dopo aver selezionato un oggetto, decidere cosa fare;
(3) dopo aver selezionato un oggetto, il sistema puo' mostrare all'utente quali sono i comandi che possono a quel punto essere invocati (evidenziando con tecniche varie sui menu i soli correntemente utilizzabili); cio':
   - riduce la quantita' di informazioni che l'utente deve memorizzare;
   - rende piu' difficile commettere errori;
   - evita di specificare tutta una serie di comunicazioni di errore nel corpo dei programmi;

Ambienti a finestre

Un ambiente **multi-window** e' caratterizzato dalla presenza contemporanea sullo schermo di piu' finestre e rende possibile rappresentare in ogni finestra lo stato di un'applicazione o di un diverso oggetto (vedi fig.1). In un contesto in cui l'utente debba gestire contemporaneamente piu' applicazioni o piu' attivita' nell'ambito della stessa applicazione, ha importanza fondamentale che tutti gli oggetti o lo stato delle applicazioni siano simultaneamente presenti e gestibili sullo schermo. La possibilita' di gestire piu' finestre sullo schermo e' fondamentale per permettere di simulare, lavorando su una workstation, il tipico tavolo di lavoro generalmente coperto da fogli, libri, o strumenti quali calcolatrice, dizionario, orologio, etc. (fig.1).

In un ambiente multi window, poiche' le varie finestre spesso contengono informazione con profondi rapporti di interrelazione, deve essere possibile estrarre informazione da una finestra e copiarla su un'altra. Il paradigma **taglia e incolla** (*cut and paste*) e' stato sviluppato appunto per soddisfare tale necessita', simulando a livello di interfaccia quello che si farebbe in genere manualmente con forbici e colla. Generalmente richiede che l'utente selezioni un oggetto (oggetto che varia nel tipo a seconda dell'applicazione associata alla finestra corrente) e che quindi scelga tra un'operazione di *taglio* (l'oggetto viene eliminato dalla finestra corrente ed inserito in una buffer area) o di *copiatura* (l'oggetto rimane invariato nella finestra, si effettua solo la copia nella buffer area). Una volta inserito nella buffer area potra' essere copiato successivamente su qualsiasi altra finestra utilizzando il comando *incolla.*

Undo

Da' la possibilita' di eliminare gli effetti di un comando, tornando alla situazione precedente all'esecuzione del comando. La presenza di tale strumento rende molto piu' sicuro il lavoro

dell'utente e aumenta la possibilita' di apprendere il funzionamento del sistema durante l'uso poiche' gli effetti negativi di eventuali errori dovuti alla scarsa perizia sono facilmente eliminabili.

Visualizzazione di menu

La semplicita' d'uso di una applicazione aumenta notevolmente ove sia possibile avere visualizzato direttamente su video l'insieme di comandi disponibili in un determinato istante. Diviene in questo modo sempre piu' difficile che l'utente cadere in quello stato in cui la mancanza mnemonica provoca continui blocchi dell'attivita' lavorativa.

Per la visualizzazione di liste o gerarchie di comandi sono ormai universalmente usati in interfacce evolute i classici "**pull-down**" menu (detti anche menu a tendina) il cui funzionamento e' illustrato in fig.2. Particolarmente valida per la visualizzazione di gerarchie di comandi risulta anche l'utilizzazione di gerarchie di "**pop-up**" menu, in cui i singoli menu di livello intermedio vengono visualizzati, sopra il precedente contenuto della finestra, solo quando il cursore punta il relativo campo nel menu di livello superiore (vedi fig.3).

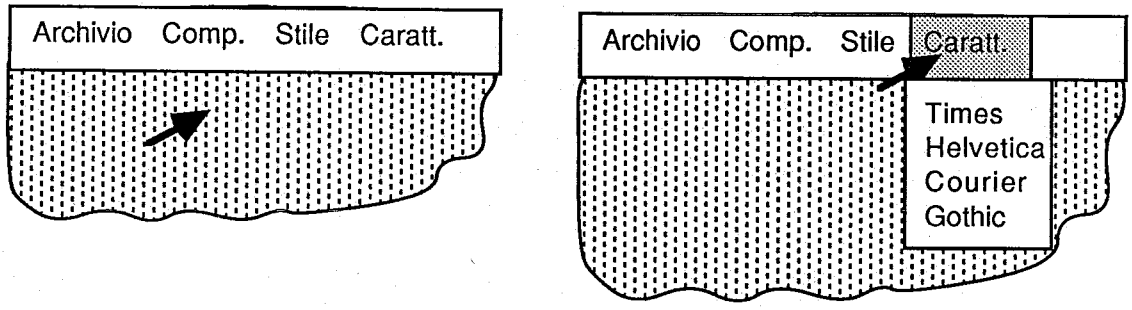

Fig.2 : Menu pull-down.

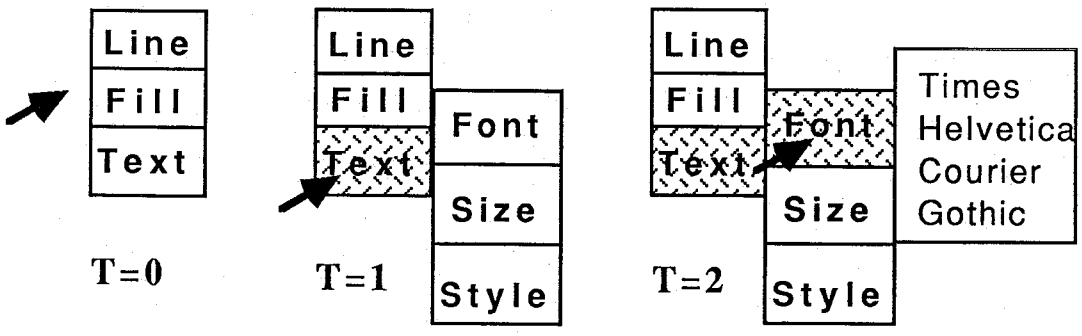

Fig.3 : Menu pop-up gerarchico.

[3] Livelli nella progettazione di interfacce

L'interazione tra utente e computer puo' essere descritta a due livelli di astrazione: a livello di "**significato**" delle informazioni che si scambiano e di "**forma**" fisica attraverso cui avviene tale attivita'. Ambedue possono a loro volta essere divise ulteriormente in sottolivelli, come descritto in fig.4.

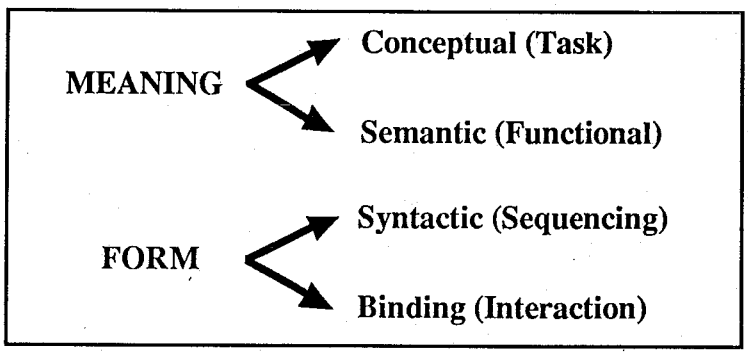

Fig. 4: Suddivisione in livelli dell'attivita' di progettazione di un'interfaccia.

Conceptual (task) **level:**

primo livello da prendere in considerazione nella definizione di una HCI; a livello concettuale viene effettuata un'analisi astratta (**task analysis**) degli **oggetti** gestiti dall'applicazione, delle loro **proprieta'**, delle **operazioni** (o azioni) che l'utente potra' richiedere su essi e delle **relazioni** che legano oggetti ed azioni. Risultato della fase di task analysis, applicata al sistema di cui si vuole progettare la user interface, e' quindi l'individuazione dell'insieme di oggetti, di relazioni, di proprieta' e di azioni.

Es. Un **File System** puo' essere descritto a livello di task analysis come insieme di:

Oggetti: file, cartelle

Azioni (sugli oggetti):
crea, apri, chiudi, salva, cancella, muovi, duplica ...

Relazioni tra oggetti:
cartelle organizzate in modo gerarchico, ogni file e' contenuto in una cartella, ...

Proprieta':
tipo dell'oggetto, data di creazione, stato (oggetto corrente, aperto/chiuso, ..)

L'analisi dei task, come si puo' notare, e' un'attivita' del tutto astratta e deve fornire l'intelaiatura concettuale su cui andra' costruito il sistema. Per tale ragione i risultati devono essere piu' vicini possibile al modo usuale di agire dell'utente tipo, lasciando il piu' possibile inalterato il modello concettuale che esso ha ereditato dall'uso di questo o di precedenti sistemi (**user conceptual model**). La descrizione astratta fornita dalla task analysis potra' in seguito condurre a differenti realizzazioni; essendo definita solo ai successivi livelli la particolare implementazione di oggetti ed azioni.

Semantic (functional) **level:**

livello in cui viene specificata dettagliatamente ogni funzionalita' dell'applicazione (ad es.: quali informazioni sono necessarie per eseguire un'operazione su un oggetto, che errori possono verificarsi e come sono trattati, quali sono i risultati di ogni azione). Si definiscono quindi in dettaglio il significato dell'operazione e le condizioni al contorno (semantica dell'operazione), ma non la sequenza di azioni elementari od i device necessari alla realizzazione della stessa (aspetti trattati nei successivi livelli).

Fig. 1 : esempio di descrizione di un'azione (FileTogether) a livello concettuale e semantico.

Conceptual

Task entities: files, folders, ..

Task goals: filing related document together

Task procedure FileTogether (type of information I, name of folder N)

```
begin
 CreateNewFolder (N);
 for each (file with information of type I) do
      move (file into new folder N)
 od
end
```

Task Method: to achieve the goal of filing related documents together, use the Procedure *FileTogether* .

Semantic

Semantic procedure FileTogether (type of information I, name of folder N)

```
begin
 User :  CreateNewFolder (N);
 System :  Show creatin of new folder with name N;
 For each file do
      user : inspect file type;
      system : report file type;
      if user : observes that file type has information of type I then
           begin
            user: move (file to folder N);
            system : represent the file moving into N;
           end
 od
end
```

Semantic Method: to achieve the goal of filing related documents together, use the Semantic Procedure *FileTogether* .

Syntactic (sequencing) **level :**

si specifica il dialogo, in termini di sequenza di singole sotto-operazioni di I/O, che utente e computer devono effettuare per portare a termine ogni azione definita a livello superiore.

Ogni singola sotto-operazione e' considerata un'operazione di I/O elementare, nel senso che non puo' essere decomposta in operazioni piu' semplici senza perdere il significato semantico che la caratterizza. Le sotto-operazioni di I/O costituiscono quindi i blocchi elementari sintattici con cui una varieta' di interazioni "grammaticamente corrette" possono essere generate.

Binding (interaction) **level:**

ogni singola unita' elementare di I/O e' decomposta nella serie di primitive HW/SW disponibili effettivamente sulla macchina. Riguardo all'input, a livello di binding si scelgono le tecniche di interazione piu' adatte alla realizzazione delle operazioni elementari descritte al livello superiore (movimenti del mouse, clicking, utilizzo della tastiera, voice inputs, etc.). Per l'output si tratta invece di definire la combinazione di primitive ed attributi che realizzano i suoni, le icone o gli altri simboli grafici utilizzati al livello sintattico. Sono quindi affrontate a livello di binding le problematiche classiche coinvolte nel collegamento di un livello device independent (sintattico) ad uno device dependent (binding).

Metodologie di progetto

La suddivisione in livelli descritta e' funzionale al metodo di definizione "top-down" di un'interfaccia. Nella progettazione top-down di una User Interface si lavora inizialmente alla task analysis a livello concettuale, valutando sia l'insieme di operazioni di base e gli obiettivi richiesti dall'utente che l'aspetto globale del sistema. Quindi si passa ad esaminare ogni singolo comando a livello funzionale, specificandone il comportamento a maggiore livello di dettaglio. Le ultime due fasi sono in genere affrontate parallelamente, specificando gli stili di interazione piu' adatti ed utilizzandoli nel tradurre nella sintassi del sistema le funzionalita' previste ai livelli superiori.

E' quindi chiaro come l'approccio top-down sia rivolto essenzialmente alla specifica, a livello di definiziomne sempre piu' basso, piuttosto che alla definizione di prototipi. Tale

tipo di approccio viene sempre piu' spesso messo in discussione a favore della metodologia **bottom-up**, che si basa su una significativa attivita' di prototipazione. nella progettazione bottom-up di user interface la fase di specifica e' il piu' possibile sostituita dall'attivita' di prototipazione in cui si cerca di implementare abbastanza da poter simulare la fase di dialogo, per verificare e validare le principali funzioni dell'interfaccia direttamente durante la progettazione, con tempi e costi piu' ridotti.

La disponibilita' di validi strumenti di supporto alla prototipazione, che permettano la semplice realizzazione di versioni prototipali, la loro trasformazione o verifica ed eventualmente la loro conversione automatica nel prodotto finale, e' naturalmente essenziale per un approccio di tipo bottom-up.

Va inoltre sottolineato come negli ambienti di supporto all'attivita' di prototipazione si faccia un uso sempre piu' massiccio della programmazione Object Oriented. Cio' sia poiche' gli ambienti di programmazione object oriented mettono spesso a disposizione un gran numero di strumenti utili alla realizzazione di interfacce, tra cui ampie librerie di tecniche di interazione, che per la loro caratteristica di essere decisamente orientati alla manipolazione diretta. Altro notevole punto di forza dei linguaggi object oriented e' il permettere la programmazione incrementale, caratteristica di indubbia utilita' nella realizzazione di prototipi, supportata dal meccanismo di definizione di sottoclassi e dall'ereditarieta' dei metodi tra classi e sottoclassi.

[4] Tool di supporto alla progettazione

Sono in genere oggetti piu' sofisticati ed evoluti delle comuni librerie grafiche, mettendo a disposizione metodi e strategie per aumentare la produttivita' nello sviluppo di interfacce ed, in forma di primitive di medio livello, gli elementi chiave delle interfacce di maggiore successo (tecniche di interazione).

I tool di supporto possono essere suddivisi in due categorie fondamentali. Gli **Interaction Technique Libraries (ITL)**, noti anche come User Interface Toolkits, sono utili al livello piu' basso della progettazione, occupandosi del binding delle tecniche di interazione (window, menu, etc) alle primitive grafiche di base. Gli **User Interface Management Systems (UIMS)** sono invece strumenti molto piu' potenti, gestendo completamente il controllo a livello di sequencing dell'intero dialogo.

Riguardo alla reale disponibilita' di questi strumenti va detto che mentre il settore degli ITL puo' ormai essere considerato sufficientemente stabilizzato, con svariate proposte commerciali di tutto rispetto, a livello di UIMS non e' possibile andare per ora al di la' delle esperienze e delle realizzazioni prototipali sviluppate in ambiente di ricerca e militare. La notevole attivita' di ricerca sia universitaria che privata lascia comunque prevedere a breve un grosso sviluppo commerciale di questi ultimi strumenti.

[4.1] Interaction Technique Libraries (ITL)

Denominati a volte anche Toolkits o Window Systems, sono generalmente costituiti da set di subroutines, ognuna delle quali relativa ad una tecnica di interazione. Si appoggiano alle capacita' di gestione dell'I/O di pacchetti grafici ed, oltre a rendere non necessaria una programmazione di basso livello, rendono possibile generare interfacce dotate di uniformita' di stile.

In fig.5 sono schematizzati le varie componenti di un ITL. A livello piu' basso il Window System e' responsabile della gestione delle primitive grafiche, della posizione del cursore, dell'ordinamento in profondita' delle finestre, dell'uso di risorse condivise come tastiera e mouse. Sulle funzionalita' del window system di base poggia la parte di piu' alto livello dell'ITL che gestisce sia la manipolazione di oggetti complessi quali menu, dialog boxes, scroll bars, che il controllo degli eventi e le interazioni con il file system.

L'architettura degli ITL varia sensibilmente in termini di indipendenza dai device, di estensibilita', di modelli scelti per la rappresentazione delle immagini e di integrazione nel

kernel del sistema operativo nativo della macchina. Una classificazione degli ITL puo' essere condotta rispetto al loro maggiore o minore collegamento a aprticolari sistemi HW/SW. Si parla infatti di sistemi **kernel based** e di sistemi basati sul modello **cliente/servente.**

Nel primo caso ci si riferisce ad ITL sviluppati per personal computer (**Toolbox** e **MacApp** in ambiente Apple, **Windows** in ambiente Ms-Dos) e caratterizzati da una alta integrazione al sistema operativo nativo, che permette di aumentare sensibilmente le prestazioni in ambienti generalmente monoutente e monotask. Il vantaggio di poetr accedere direttamente a routine di kernel del sistema operativo, in genere memorizzate su ROM e appositamente estese per potenziatre le possibilita' di gestione di I/O come nell'architettura Macintosh, comporta pero' una corrispondente perdita in termini di device-independence, rendendo spesso nulla la portabilita' su altre architetture del SW sviluppato.

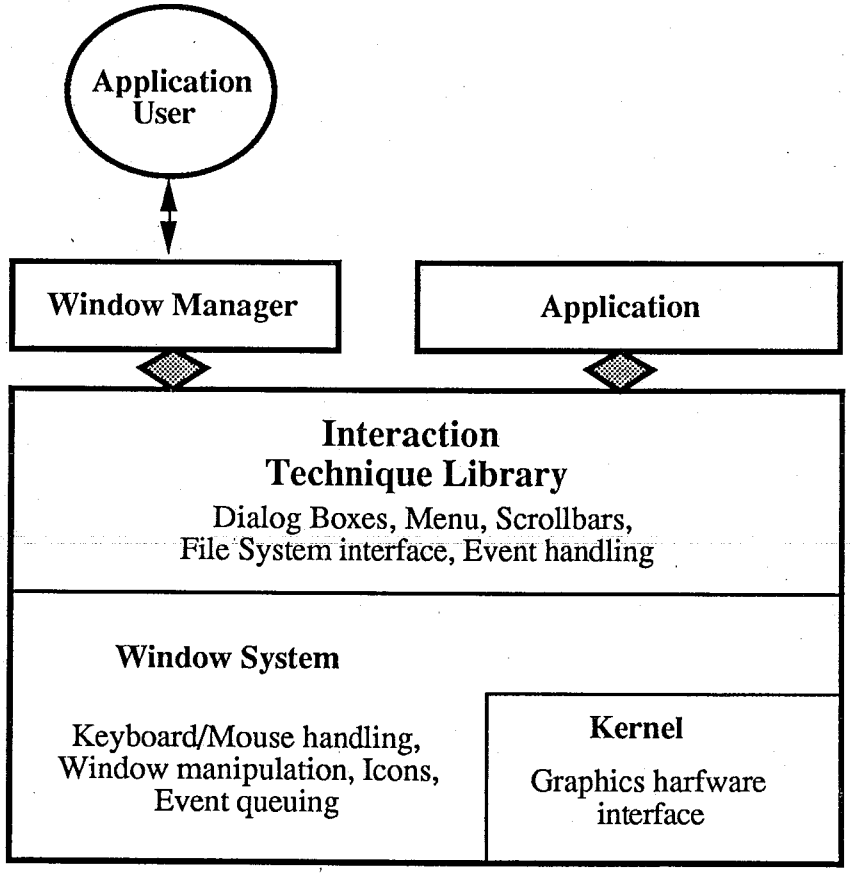

Fig.5: Architettura di una Interaction Technique Library

L'approccio clienti-serventi e' stato sviluppato per macchine di fascia piu' alta (in genere workstation Unix) e prevede una gestione delle richieste basata sull'interazione tra piu' processi cooperanti. Piu' processi clienti, corrispondenti a diverse applicazioni, richiedono l'esecuzione di funzioni ad uno o piu' processi serventi, ove ogni servente e' in

genere associato ad una stazione di I/O. Dal meccanismo di chiamata procedurale si passa quindi ad un protocollo di comunicazione basato sullo scambio di messaggi tra processi, effettuabile anche in rete. Questo approccio, scelto da ITL quali X-Windows, lo standard de facto sviluppato dal M.I.T., e NeWS della SUN, probabilmente l'ITL piu' promettente a livello di macchine Unix, garantisce un'alta indipendenza dai device ed un'ampia portabilita' del SW.

Va comunque fatto notare che gli ITL kernel-based mettono per ora a disposizione primitive piu' sofisticate: ITL client-server spesso non gestiscono lo strato di funzionalita' di livello alto della fig.5. Inoltre la gestione delle comunicazione tra processi puo' causare un grosso overhead con possibili sensibili rallentamenti delle prestazioni. Va infine sottolineato come questi strumenti, fatta eccezione per il solo MacApp della Apple, risultino di difficile uso ed in genere adatti solo a programmatori molto esperti; cio' sia a causa del grande numero di funzioni messe a disposizione che delle difficolta' connesse alla gestione di un ambiente programmativo non classico, con elaborazioni caratterizzate dal complesso controllo degli eventi.

[4.2] User Interface Management Systems (UIMS)

Un UIMS non fornisce allo sviluppatore solo una libreria di primitive da usare nel contesto di un linguaggio di programmazione: sebbene le possibilita' offerte dagli UIMS varino notevolmente, una loro caratteristica essenziale e' la possibilita' di poter specificare la "dialogue sequence".

In fig.6 e' mostrata l'organizzazione generale di un UIMS.

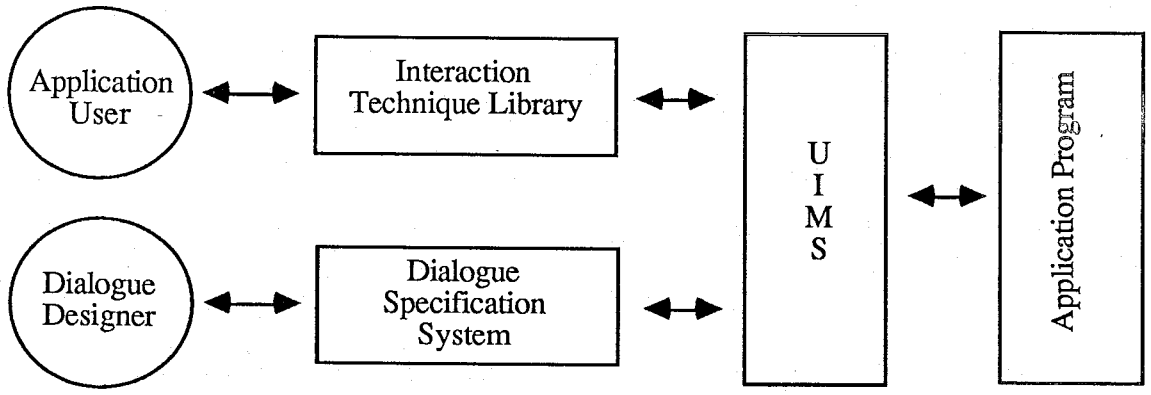

Fig.6: Interazione tra moduli costituenti un UIMS.

Per specifica della **Dialogue Sequence** si intende la possibilita' di specificare in modo assistito la sequenza di azioni dell'utente e le risposte dell'interfaccia e dell'applicazione. Per tale specifica sono utilizzati vari formalismi, tra cui diagrammi di stati (transition networks), event processing languages e grammatiche in B.N.F. In tutti questi formalismi e' presente il concetto base di "stato" dell'interfaccia, rappresentato da un insieme di variabili di stato. Le tecniche per la specifica del dialogo danno la possibilita' di descrivere il comportamento del sistema in risposta ad ogni azione dell'utente; la conseguente risposta del sistema e' funzione dello stato corrente e puo' manifestarsi nella chiamata di una o piu' action routines, nella modifica dello stato e nell'abilitazione, disabilitazione o modifica della tecnica di interazione che governera' la prossima azione dell'utente.

Nel caso che lo stato dell'interfaccia possa essere codificato da un'unica variabile con un modesto numero di valori, sono utilizzabili **transition networks** (diagrammi di stati). Ogni stato del diagramma corrisponde in questo caso ad un stato del sistema, e generalmente una "action routine" e' associata ad ogni transizione (fig.7).

Un chiaro svantaggio dei T.N. e' la scomodita' di mappare tutti i possibili valori delle variabili di stato di un'interfaccia in un'unica variabile; inoltre le routine non possono in questo formalismo modificare lo stato, la transizione di stato dipende solo dall'azione dell'utente. Tali limiti sono superati dagli **Augmented Recursive Transition Network** (ARNT), che permettono sia di contenere il numero degli stati per mezzo della specifica di sotto diagrammi, richiamabili da altri stati con modalita' simile alla chiamata procedurale, che la modifica del valore delle variabili di stato da parte delle action routine. Quest'ultima caratteristica e' fondamentale per la definizione della casistica di errore: nel caso che durante l'esecuzione di una routine si verifichi un errore, e' possibile in questo caso effettuare una transizione in uno stato di errore diverso da quello in cui sarebbe terminata normalmente la routine.

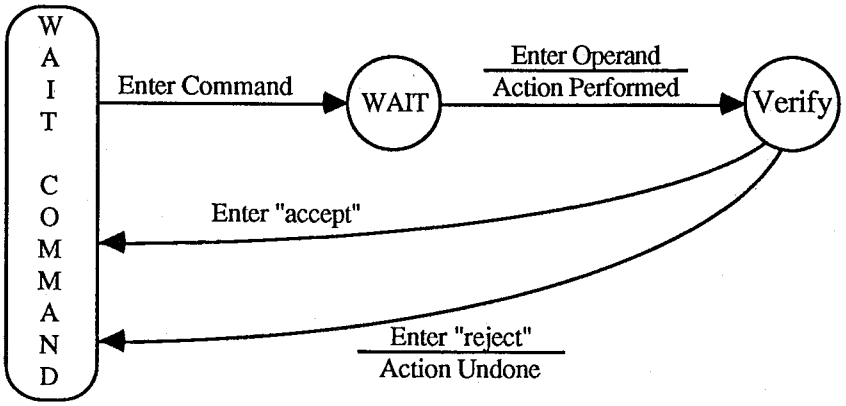

Fig 7 : un esempio di Transition Network .

Una delle caratteristiche positive degli A.T.N. e' la loro maggiore comprensibilita', per utenti del tutto non informatizzati, rispetto ai linguaggi . Campi in cui gli ATN trovano largo uso sono sia quello della specifica a basso livello delle tecniche di interazione che quello ad alto livello di astrazione della specifica delle transizioni tra componenti fondamentali dell'applicazione interattiva.

La **Bakus Naur Form** e' notevolmente deficitaria in termini di semplicita' di comprensione; e' quindi usata abbastanza raramente.

All'aumentare della complessita' del dialogo da formalizzare risulta preferibile l'uso di event languages (in fig.8 un esempio di **event language** specializzato alla specifica del dialogo), considerati gli strumenti piu' potenti per specificare la sequenzialita' e la molteplicita' di condizioni associati a sistemi a transizioni. argomento attuale di ricerca e' la definizione di linguaggi iconici, basati sull'uso di rappresentazioni sintattiche non testuali, che potrebbero accoppiare alla potenza descrittiva dei linguaggi quella autoesplicativa degli ambienti iconici.

Fig.8 : esempio di event language per la specifica del dialogo.

An example of a specialized event language for dialogue specification which illustrates these concepts is:

```
If UserAction = ObjectSelection and SelectedObjectType = Curve then
   begin
      CSO = true                                   {means there is a currently selected object}
      enable (Reshape, Cut, Copy)                  {system routine to enable commands -
                                                    these are commands which become enabled
                                                    when a curved object has been selected}
      highlight(SelectedObjectIdentification)      {system routine to highlight an object}
   end

if UserAction = CommandSelection and Command = Copy then
   call CopyObject(SelectedObjectIdentification)   {call an action routine to do the
                                                    copying}
   enable(Paste)                                   {Now that have copied an object, paste can be
                                                    enabled}

If UserAction = CommandSelection and Command = DrawLine then
   begin
      CSO = false                                  {any previously selected object is deselected}
      disable(Delete, Cut, Copy, Reshape)          {No CSO, so disable commands on objects}
      call DrawLine                                {procedure which will draw line. If user actually draws a
                                                    line, DrawLine will set CSO to true, SelectedObjectType,
                                                    to Line, and SelectedObjectIdentification to the ID of the
                                                    new line. If user aborts line drawing, variables will not
                                                    be set.}
      If CSO = true then enable(Delete, Cut, Copy)
   end
```

Altre funzionalita' degli UIMS

- **Editori di elementi grafici:** editori interattivi che permetto di creare in modo semplice tutti quegli elementi grafici che verranno utilizzati nell'interfaccia (organizzazione dello schermo, icone, menu, bordi delle finestre, ...). Tali editori possono essere presenti anche negli ITL.

- **Strumenti per la definizione di HELP/PROMPT:** danno la possibilita' all'utente di specificare testi/elementi grafici da visualizzare in corrispondenza di richieste di help o di prompt di sistema. Da notare come tali informazioni siano generalmente 'context sensitive', cioe' dipendano strettamente dallo stato del sistema.

- **File di messaggi** su file esterni ai programmi dell'applicazione.

- **Supporto per la funzione di UNDO:** stack che memorizzano per ogni stato corrente la serie di azioni precedentemente effettuate e di controazioni che ne rendano possibile l'UNDO.

- **User profile:** vengono utilizzati per modificare il modo in cui l'interfaccia si manifesta in funzione del tipo di utente che la utilizza (ad es. organizzazione dello schermo, menu destri/mancini, velocita' dei device di I/O, uso del colore, livello di aiuto, ...).

[5] STATO DELL'ARTE

Ricerca :

- ITL ==> campo ormai stabilizzato

- UIMS ==> sufficientemente definiti, esistono realizzazioni sviluppate in ambito accademico

- il grosso delle attivita' e' indirizzato alla definizione di tools che aiutino nelle prime due fasi della progettazione, i livelli concettuale e semantico

Industria:

- ITL ==> sono disponibili varie realizzazioni; si sta imponendo uno standard de facto (X Windows)

- UIMS ==> in via di sviluppo

- Concettuale/Semantico ==> carta, penna e cervello.

[6] Attivita' in corso alla George Washington University (Prof. James D. Foley)

Il progetto di ricerca in corso e' indirizzato allo sviluppo di una tecnica di rappresentazione formale utilizzabile nella fase di sviluppo a livello concettuale e funzionale di una interfaccia.

La rappresentazione scelta incorpora informazioni riguardanti oggetti, azioni, attributi, pre- e post-condizioni associate alle azioni. E' in corso di sviluppo un sistema interattivo, basato su un frame-based expert system shell, da utilizzare come strumento per la specifica di interfacce.

Il sistema mettera' a disposizione dello sviluppatore, una volta specificata una prima ipotesi di interfaccia, le seguenti funzionalita':

- applicazione automastica alla specifica di controlli di completezza e consistenza;
- trasformazione della specifica in una serie di specifiche funzionalmente equivalenti, ognuna delle quali fornisce differente user interface per la realizzazione della stessa funzionalita';
- valutazione della velocita' di uso di tali prototipi (definiti solo a livello di specifica) sulla base di uno scenario di task tipici;
- possibilita' di fornire la specifica prescelta come input di un UIMS, per una implementazione il piu' possibile automatica.

In fig.9a viene descritto, nel formalismo scelto dal gruppo di Foley, un semplice sistema per la gestione di grafica 2D. In fig.9b e' invece illustrato l'algoritmo che permette di fattorizzare un attributo (cioe' trasformare da esplicito in implicito un attributo, eliminandolo dal corpo di chiamata delle azioni) in modo del tutto automatico, una volta definito l'algoritmo di trasformazione; in fig.9c la nuova specifica dopo l'applicazione dell'algoritmo.

Fig. 6a : Formalizzazione, a livello semantico, di un semplice sistema per la gestione di grafica 2D.

```
base_design: begin
{declarations of the types of objects on which the user interface commands operate}
{shape is an object class with two subclasses. The actions and attributes originate with this
class, and can be inherited by object subclasses of shape.}

type shape
   superclasses   : ()
   subclasses     : (triangle, square)
   actions        : (create_shape, delete_shape, rotate_shape) : originates, heritable
   attributes     : (position, color) : originates, heritable
   relations      : ()

{two subclasses of the object class shape. Actions and attributes are inherited from the
shape superclass, and cannot be inherited by subclasses because there are none.}

triangle, square
   superclasses   : (shape)
   subclasses     : ()
   actions        : (create_shape, delete_shape, rotate_shape) : inherits from
                    shape, not heritable
   attributes     : (position, color) : inherits from shape, not heritable
   relations      : ()

{declarations of attributes of the above objects}

position   : range [0..10] x [0..10]
color      : set (1, 1) of (red, green, blue, white, black, cyan, magenta, yellow)
angle      : range [0..360]

{initial values for any pre-conditions}

initial: shape_objects = 0

{operations on the objects, with pre-conditions and post-conditions}

create_shape(p:position, c:color, a:angle, type_of_shape:shape_class)
post-condition: shape_objects = shape_objects + 1

pre-condition: shape_objects <> 0
rotate_shape(obj:shape, a:angle)

pre-condition: shape_objects <> 0
delete_shape(obj:shape)
post-condition: shape_objects = shape_objects - 1

end {base_design}
```

Fig.6b : Algoritmo di trasformazione per la fattorizzazione di un attributo.

The transformation algorithm which factors an attribute is:

FactorAttribute(*factored_attribute_type*)

Add an intial condition
initial: *factored_attribute_type*_set = **true**

Add an action
set_*factored_attribute_type*(x: *factored_attribute_type*)
post-condition: *factored_attribute_type*_set = **true**

For all actions with *factored_attribute_type*
Change type of *factored_attribute_type* to **implicit**
Add a precondition
pre-condition: *factored_attribute_type*_set = **true**

Fig.6c : Il sistema della fig.5a dopo l'applicazione del precedente algoritmo.

```
factored_design: begin
initial: shape_objects = 0                    {initial values for post conditions}
initial: color_set = false                    [added automatically]
set_color(c:color)                            [added automatically]
post-condition: color_set = true              [added automatically]
pre-condition: color_set = true               [added automatically]
create_shape(p:position, c:color implicit, a:angle, type_of_shape:shape)
                                              [implicit added automatically]
post-condition: shape_objects = shape_objects + 1
pre-condition: shape_objects <> 0
rotate_shape(obj:shape, a:angle)
pre-condition: shape_objects <> 0
delete_shape(obj:shape)
post-condition: shape_objects = shape_objects - 1
end {factored_design}
```

[7] Un sistema ITL : X Windows

Il sistema X Windows e' stato sviluppato nell'ambito di due diversi progetti di ricerca avviati al M.I.T. nel 1984, progetti caratterizzati entrambi dalla necessita' di una sofisticata gestione dello schermo e dei dati in esso visualizzati. X Windows (in seguito X), la cui prima specifica va considerata un effetto laterale degli studi affrontati in questi due progetti, e' stato successivamente sviluppato in modo autonomo ed ha raggiunto una larga diffusione soprattutto in ambiente Unix, caratterizzandosi come standard "de facto" nel campo degli Interface Technique Libraries. E' distribuito gratuitamente dal M.I.T.

Una caratteristica peculiare di X e' quella di essere definito su un protocollo di rete: la comunicazione asincrona tra processi sostituisce il piu' tradizionale meccanismo di chiamata procedurale o chiamata di funzioni di nucleo.

[7.1] Requisiti di un ITL, messi in evidenza nello sviluppo di X Windows

Una user interface puo' essere vista come l'unione di due principali componenti: la **application interface**, che governa l'interazione tra l'utente e gli elementi grafici propri di ogni applicazione, e la **management interface**, che governa l'interazione globale tra l'utente e la serie di meccanismi di input e di finestre presenti su display. Quest'ultima governa il modo in cui le varie applicazioni sono visualizzate sullo schermo e in cui l'utente passa da un'applicazione all'altra.

Nella progettazione di X sono stati evidenziati i requisiti di un window system di base, cioe' del sistema di supporto per la definizione sia di application che management interfaces:

1) deve essere facilmente realizzabile su un ampio numero di display;
2) deve essere interfacciabile con applicazioni device-independent;
3) deve essere network-transparent; per dare la possibilita' alle applicazioni di utilizzare in modo trasparente device posti su nodi diversi;
4) deve permettere la visualizzazione e gestione contemporanea di piu' applicazioni (simulazione del tavolo di lavoro);
5) deve permettere la generazione di svariati tipi di management ed application interface; il sistema deve quindi mettere a disposizione "strumenti" per la definizione di interfacce e non "politiche" predefinite;
6) deve permettere la sovrapposizione tra finestre, mantenendo la possibilita' di modificare quelle parzialmente coperte;

7) deve gestire gerarchie di finestre (di dimensioni qualsiasi, definibili anche dinamicamente), ed un'applicazione deve essere in grado di poter usare piu' finestre contemporaneamente;
8) deve gestire in modo estremamente efficiente testi, grafica 2D e immagini;
9) deve essere il piu' possibile estensibile.

[7.2] Struttura del sistema

Il sistema X e' basato su un modello client-server (per i requisiti 2 e 3). Un processo **server** di controllo e' attivo per ogni display; un'applicazione, a cui corrisponde un processo **client**, comunica col server utilizzando la comunicazione tra processi se entrambi sono situati sullo stesso nodo, od una connessione tramite rete in caso contrario. Molti clienti possono accedere allo stesso server ed un cliente puo' essere connesso a piu' server contemporaneamente. Compito essenziale del server e' quello di gestire le varie richieste di accesso in output allo schermo generate dai client, e di dirigere gli input dell'utente ai client destinatari (vedi fig. 10).

Il server incapsula il sistema di finestre di base ed i vari strumenti necessari alla realizzazione di varie tecniche di interazione.

Tutti gli aspetti device-dependent sono confinati a livello di server, mentre il protocollo di comunicazione tra clienti e server e' totalmente device-independent. Inoltre, anche lo stesso server e' definito in gran parte in forma device-independent, sulla base di un limitato nucleo device-dependent.

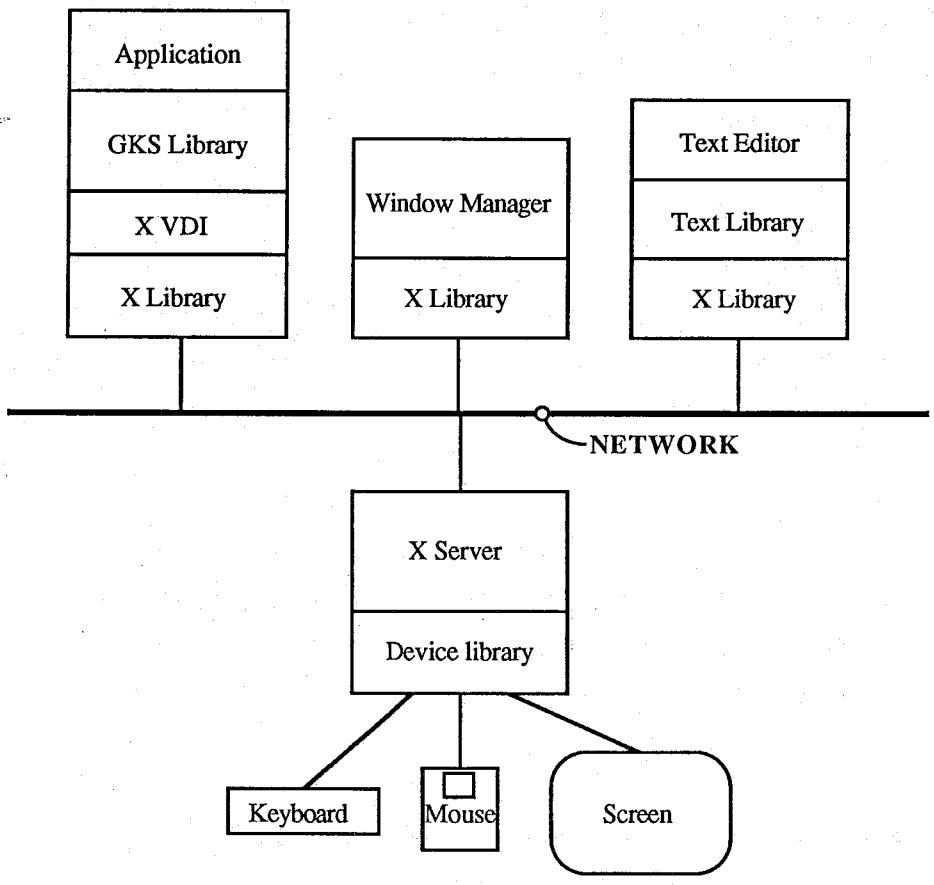

Fig.10 : Struttura del sistema X Windows

[7.3] Gerarchia di finestre

Il sistema X Window gestisce gerarchie di finestre rettangolari, oguna delle quali associata ad un server. La finestra radice (root), copre l'intero schermo; le top-level windows di ogni applicazione sono create come sottofinestre della root window.

Le sottofinestre di una gerarchia possono sovrapporsi in ogni modo; sottoalberi diversi della stessa gerarchia non possono pero' coprirsi a vicenda: se w1 del sottoalbero S1 copre w2 di S2, nessuna delle sottofinestre in S2 potra' coprire finestre in S1.

Una finestra non e' vincolata per dimensioni e posizionamento da quelli della finestra che la precede nella gerarchia; di una finestra figlio sara' pero' sempre visibile solo la parte contenuta nello spazio coperto da quella che la precede nella gerarchia (le eventuali parti esterne sono clippate sul contorno della finestra padre).

Le finestre possono avere bordi, il cui shading e' effettuato direttamente dal server.

Una window puo' essere:

- unmapped ==> non visibile sullo schermo
- mapped ==> visibile sullo schermo solo se anche le precedenti nella gerarchia sono mapped

Output sulle window:

- l'output su finestre foglia viene sempre clippato sui bordi della finestra;
- l'output su una finestra intermedia puo' essere clippato sui bordi delle finestre figlio precedentemente mappate (clipped mode) o puo' essere visualizzato sull'intera estensione della finestra (draw-through mode).

Sistemi di coordinate :

ogni finestra ha il suo proprio sistema di coordinate (origine = upper left corner), in modo che i comandi di output siano definibili nello spazio della finestra e non in quello di dispositivo. Cio' e' fondamentale per la device-indepependence degli applicativi.

I sistemi di coordinate sono in genere definiti come spazi 2D discreti.

Operazioni elementari:

- **create** : una finestra viene creata (unmapped) specificando finestra padre, posizione all'interno della finestra padre, dimensioni (due assi);
- **destroy** : la distruzione di una finestra distrugge tutto il sottoalbero che parte da questa;
- **map/unmap;**
- **move** : per lo spostamento delle finestre sullo schermo;
- **resize** : per variare le dimensioni;
- **visualizzazione in primo piano** : per visualizzare l'intera finestra, anche se sono presenti suoi discendenti mapped.

La progettazione del server e' stata espressamente orientata a rendere l'uso delle finestre molto efficiente; cio' allo scopo di rendere possibile l'uso delle finestre per la realizzazione di singole voci di menu, bottoni virtuali, etc. Per permettere utilizzazioni di questo tipo il server deve essere in grado di gestire simultaneamente anche varie decine di finestre, e quindi la velocita' di gestione risulta un requisito fondamentale.

[7.4] Gestione del colore

X vede il display come matrice bidimensionale, con pixel ad n-bit associati ad ogni coppia di coordinate intere. Per ragioni di device-indep. i valori colore dei pixel non sono codificati direttamente a livello di interfaccia, ma il server gestisce sue color map.

C'e' la possibilita' di gestire grafica in overlay, utilizzando parte dei bit colore.

[7.5] Primitive grafiche e testi

X mette a disposizione operazioni grafiche di livello non elementare: linee, rettangoli, curve e varie fonti per la visualizzazione di testi. In cio' X si differenzia profondamente da altri sistemi che utilizzano primitive di basso livello, essenzialmente lettura e scrittura di pixels. Basare le operazioni grafiche sulla scrittura/lettura di pixels assicura buoni livelli di performance solo se la memoria display puo' essere manipolata direttamente dalle applicazioni; poiche' l'accesso diretto alla memoria display non e' sempre possibile, specialmente in ambiente distribuito, la filosofia scelta nella specifica di X e' l'unica che garantisce indipendenza dall'HW e buone performance.

X Windows fornisce sia **bitmaps** (aree rettangolari, single bit per pixel) che **pixmaps** (aree rettangolari definite su N piani di memoria). Sono create trasmettendo il valore dei bit al server e possono essere di ampiezza qualsiasi.

Le bitmaps sono comunemente usate come maschere per region clipping e per la realizzazione di cursori. Le pixmaps sono soprattutto utilizzate come tiles (mattonelle) per la realizzazione di pattern nel riempimento di aree.

Primitive grafiche

- riempimento di una finestra con un valore colore;
- riempimento di una finestra utilizzando un pattern particolare;
- visualizzazione di immagini ad due o piu' livelli livelli di colore;
- CopyArea : per spostare parti rettangolari di una finestra all'interno della stessa finestra (classica operazione "bitblt", bit block transfer);
- disegno di linee, sia rette che curve; si possono definire sia spezzate aperte che chiuse. Gli attributi utilizzabili per modificare l'aspetto delle linee sono molto sofisticati: ampiezza del tratto, tratteggio, utilizzazione di pattern per il tracciamento del tratto, variazione del tipo di "pennello" utilizzato in tracciamento. Inoltre, le aree chiuse possono essere riempite con pattern qualsiasi.

Testi

Per ragioni di performance, sono rese disponibili direttamente a livello di server diverse fonti rappresentate per mezzo di bitmaps (256 per ogni fonte). Tali fonti possono essere visualizzate sia in modo "mask" (sono visualizzati nella finestra solo i pixel ad 1 della

bitmap, il colore intorno alla sagoma del carattere rimane quello di sfondo della finestra) che in modo "source" (viene visualizzata l'intera bitmap).

[7.6] Gestione delle finestre

La funzione primaria di un window manager e' la riconfigurazione della serie di top-window visualizzate (variazione delle dimensioni, della posizione sullo schermo e della posizione sullo stack di visibilita'). Si possono definire due categorie di window manager: **manuali**, ove siano richiesti specifici comandi di riconfigurazione dell'utente, ed **automatici**, ove la riconfigurazione delle top-window e' gestita in modo indipendente e trasparente all'utente .

X lavora in modo manuale: i client sono infatti responsabili del posizionamento e dimensionamento iniziale delle top-windows. Le operazioni di modifica delle finestre sono in genere governate dall'utente utilizzando il mouse (es. click su una finestra per renderla corrente).

La trasformazione di una finestra in un'icona che ne simboleggia il contenuto e' una delle piu' importanti operazioni di ridimensionamento. In X e' molto semplice gestire tale trasformazione poiche', essendo le icone realizzate come finestre, e' sufficiente un unmapping della finestra e un mapping di quella relativa all'icona. L'associazione icona-finestra e' mantenuta direttamente dal server.

L'attivita' di un window manager genera continuamente la necessita' di "ripristinare" il contenuto delle finestre che, dopo essere state parzialmente/completamente coperta da altre, ritornino visibili. X Windows assegna la responsabilita' della gestione del ripristino al cliente che ha originato la finestra e non al server.

Per ripristinare il contenuto di una finestra il server dovrebbe avere informazione completa relativo al contenuto della stessa; dovrebbe quindi mantenere o una display list, contenente tutte le operazioni di output precedentemente richieste dal client sulla finestra, o una immagine offscreen del contenuto pittorico della finestra. Ambedue le soluzioni risultano di difficile gestione. Nel primo caso, o si mantiene l'intera display list (tutti i gli output effettuati dal momento della creazione della finestra), aumentando molto i tempi di visualizzazione (si visualizza ad ogni ripristino tutta la storia della finestra), oppure si e' costretti a definire tecniche molto sofisticate per la gestione di display list "aggiornate". Nel secondo caso, puo' risultare difficile garantire la disponibilita' della memoria necessaria alla memorizzazione offscreen delle finestre temporaneamente coperte (ad es. per salvare una sola finestra ampia come tutto lo schermo e' necessario memorizzare 1Mpixel).

Il ripristino a cura del client e' inoltre scelto poiche' spesso molte applicazioni, utilizzando strutture dati interne, riescono ad effettuare redisplay in modo molto efficiente.

Lo stesso approccio visto per il ripristino di finestre coperte vale in X anche per gestire il ripristino dovuto alla modifica delle dimensioni delle finestre.

Nel caso del ripristino di finestre molto complesse, che quindi richiedono tempi nettamente avvertibili dall'utente, e' buona norma che ogni client definisca uno sfondo (background) associato alla finestra. X gestisce autonomamente la visualizzazione del background per il quanto di tempo necessario ad effettuare il ripristino.

In alcuni casi, come ad esempio la gestione dei pop-up menu, il client puo' governare il ripristino salvando il contenuto della zona coperta dal menu in un'area offscreen; l'uso dell'area offscreen e' in questo caso preferito alle normali tecniche per ragioni di performance (massima velocita' di gestione).

X comunque non preclude la possibilita' che per particolari applicazioni sia il server, opportunamente programmato, a mantenere informazioni di ripristino delle finestre.

[7.7] Input

X Windows gestisce dati di input generati da mouse o da tastiera.

Sara' possibile gestire vari tipi di input event. Gli eventi sono associati alle finestre: ogni evento di input e' associato alla finestra corrente; se il client che controlla la finestra corrente non ha richiesto un input di quel tipo, il server non gli trasmette l'evento e passa a considerare la finestra (ed il relativo client) di livello direttamente superiore nella gerarchia.

Il mouse

Ogni evento proveniente dal mouse contiene: coordinate correnti, lo stato corrente di bottoni o chiavi associate al mouse ed un timestamp che permette di decidere il momento di generazione dell'evento (ad es. per decidere quando si e' in presenza di un doppio click).

Il cursore che evidenzia sullo schermo la posizione del mouse puo' avere qualsiasi forma (definibile dall'utente come bitmap).

Si dice che una finestra "contiene" il mouse se il cursore e' contenuto in parti visibili della stessa o di suoi successori nella gerarchia. Si dice invece che il mouse e' "in una finestra" se e' contenuto in una parte visibile di questa ma non in parti visibili dei successori.

Il client puo' ricevere un evento ogni volta che il mouse entra o esce da una finestra; cio' e' particolarmente utile per la realizzazione di menu.

La tastiera

Un client puo' ricevere selettivamente eventi dalla tastiera ogni volta che l'utente preme o rilascia un tasto. Gli eventi provenienti dalla tastiera non sono riportati in codifica ASCII, ma attraverso codici associati ad ogni tasto; e' il client a trasformare la codifica dei tasti in caratteri.

In X e' possibile utilizzare due modalita' di gestione dell'input da tastiera: **real-estate** e **listener**. In real-estate mode l'input proveniente dalla tastiera "segue" il mouse: e' associato alla finestra in cui e' correntemente il mouse (modo default). In listener mode invece l'input da tastiera e' associato ad una finestra, la "focus window", in modo indipendente dalla posizione del mouse. E' possibile passare da un modo all'altro in modo dinamico.

[8] Macintosh TOOLBOX

• Estremamente device dependent poiche' hard coded nel mac kernel.

• Primitive accessibili dai ling. di programmazione, con meccanismo di chiamata molto rapido.

Mette a disposizione:

- pull-down menu
- windowing
- mouse
- icone
- box di dialogo

Concetti di base:

- Applicazioni, documenti e viste:

in ogni istante 1 applicazione attiva, piu' documenti aperti per ogni applicazione, piu' viste dello stesso documento.

- Comandi object-based
- Finder:
  supervisore iconico basato sulla metafora della scrivania
- Finestre

- Selezioni:
  fondamentale poterle gestire per realizzare comandi object-based

- Fonts

Dipendenze tra:

Applicazioni, Documenti e Viste

Gestione delle finestre

- In ogni istante una sola finestra attiva
- la finestra attiva non e' mai oscurata da altre finestre
- l'interazione avviene solo con la finestra attiva
- cliccando una finestra la si fa divenire attiva
- il contenuto delle finestre non attive, se visibile, viene sempre aggiornato

Dialog Window:

- finestre utilizzate solo per la specifica di parametri;
- possono contenere 4 meccanismi di controllo o interazione:
  - bottoni
  - check box
  - radio button
  - dials

Selezioni:

- molti comandi agiscono solo sugli oggetti selezionati
- varie modalita' di gestione delle operazioni di selezione

Menu':

- Gestisce menu pull-down
- i **menu standard** (mela, file, edit) sono disponibili da tutte le applicazioni ⩵⩵⩾ lo stesso vale per **undo** e **clipboard**

[8] LINGUAGGI OBJECT ORIENTED

Ling Classici — Oggetti (variabili,strutture dati)
Procedure

Ling Object Or. — Oggetti:{Informazioni,azioni}
Messaggi

- Evoluzione dei meccanismi di astrazione:

Data Hiding (strutture dati e procedure nello stesso modulo)

==>

Data Abstraction (tipi di dati astratti, dati e procedure che li gestiscono nello stesso contesto)

==>

Object Orientation

Concetti di base dei ling. Obj.Or.:

a) **Classi** di oggetti (==> dichiaraz. di tipo)

Istanze (==> dichiar. di variabile)

[≡≡> Data abstraction]

[≡≡> Data o object identity : **l'esistenza di un oggetto non dipende dal suo valore o dalla "vita" della procedura che ne fa uso**]

b) **Variabili di istanza** :

- memoria privata di un oggetto
- contenuto non visibile all'esterno
- modificabile o accedibile solo per mezzo delle azioni (**metodi**) invocabili tramite messaggi

[≡≡> Data hiding]

c) **Metodi**:

- azioni eseguite in corrispondenza ai messaggi

d) **Ereditarieta'**:

- possibilita' di definire sottoclassi, che ereditano sia le variabili che i metodi della classe da cui derivano

- messaggi diretti ad una sottoclasse possono essere trasmessi alla classe di livello superiore e da questa eseguiti

SMALLTALK-80

- Ling sviluppato alla Xerox alla fine dei '70

- Ling. INTERPRETATO ==> basse prestazioni !!

- richiede HW :

  32 bit
  HDisk vari Mbyte
  Memoria Virtuale

- Adatto per sola prototipazione

LINGUAGGI IBRIDI

Forniscono le caratteristiche dei ling Obj.Or. come estensione di comuni linguaggi procedurali.

==>

- sono in genere compilati
- piu' efficienti
- di piu' facile approccio e sintassi (es. non costringono a pensare anche le operazioni aritmetiche con mentalita' ObjOr)
- richiedono ambienti HW meno sofisticati

- disponibili:

  Object Pascal (Mac, 150$)

  Object C (Unix,VMS)

  C++ (Bell Lab, Unix)

Perche' usare ling. Obj.Or nella progettazione di interfacce ?

- Gli ambienti di programmazione **object-oriented** mettono a disposizione un gran numero di strumenti utili nella realizzazione di user interface, tra cui ampie librerie di tecniche di interazione.

- Sono inoltre decisamente orientati alla rappresentazione della manipolazione diretta.

==>

Buon paradigma per la progettazione o prototipazione di interfacce

- Oggetti: si possono pensare come entita' autonome capaci di realizzare azioni predefinite in risposta a messaggi.

==>

analogo al modo di funzionare di un'interfaccia

- Progr. Obj. Or. permette la **progettazione incrementale** (ereditarieta' tra classi) cosi' come richiesto dalle modalita' di progettazione di Interf.

MAC APP

• Permette di realizzare interfacce sfruttando gli strumenti messi a disposizione dal Mac Tooolbox senza richiederne una conoscenza molto approfondita.

• Basato su un approccio **Object Oriented**

ES. quando viene attivata una finestra occorre:

- disegnare il contorno
- disegnare i controlli
- disegnare il contenuto
- evidenziare la selezione corrente

Approccio PROCEDURALE:

le fasi precedenti devono essere riscritte per ogni tipo di finestra e per ogni diverso tipo di contenuto

==>

Bassa riusabilita' del SW, alti tempi di implementazione.

Approccio Obj.Or. (Mac App):

a) sono messi a disposizione una serie di **classi** e di **metodi standard** che l'utente puo' utilizzare cosi' come sono oppure

b) l'utente puo' **specializzare** a piacere le classi standard, definendo apposite sottoclassi e specializzandone i metodi in funzione delle particolari esigenze.

Funzionalita' di Mac App:

- gestisce gli eventi
- gestisce la restituzione grafica
- gestisce le finestre
- gestisce i menu e gli "alerts"
- facilita l'introduzione di tecniche di "undo"
- gestisce autonomamente tutti i comandi generali (save, apri file, chiudi f., cancella, copia, etc.)

E' una applicazione che gira su :

MPW (Mac Programmers Workshop), che fornisce un ambiente operativo simil-Unix;

Mac **Toolbox**, toolkit per grafica ed interfacciamento;

Pascal Obj. Or. Apple, ling. Programm.

Costo: circa L. 650.000 (MPW, Pascal e Mac App)

[10] SISTEMI DI INTERAZIONE MULTIMEDIALE

- **Hypertext**

sistemi che consentono l'accesso in modo non lineare ad informazioni di natura testuale

- **Hypermedia**

sistemi in cui le informazioni possono essere di qualsiasi natura (testi, disegni, immagini, suoni, ...)

Storia:

Ricerca partita alla finbe dei 70 (Xerox)

Prodotti commerciali nel '87 (Hypercard)

CARATTERISTICHE DEGLI HYPERMEDIA

UNITA':

- Informazioni raggruppate in **unita'** di piccole dimensioni (cards, frames, nodes, ..) memorizzate in un database
- le unita' possono contenere informazioni sotto qualunque forma (**multimedia**)
- ogni unita' e' visualizzata in una **finestra** dedicata, capace di effettuare le operazioni standard

LINKS:

- unita' interconnesse da collegamenti (**links**) presenti nel database e visualizzabili
- i link possono formare una **rete** e non sono ristretti ad una gerarchia
- ogni nodo puo' avere link multipli entranti/uscenti
- l'utente puo' navigare nel database attivando i link che desidera
- attivazione dei link supportata HW ⟹ velocita'
- l'utente puo' creare e modificare sia le unita' che i links (**authoring**)
- possibile esaminare la struttura generale del database mediante browser.

APPLICAZIONI

- interfacce utente verso sistemi hw/sw (sistemi op., sistemi esperti, videodischi,...)
- interfacce utente verso database
- generazione documenti non lineari
- sistemi help in linea
- **sviluppo prototipi di interfacce**

BENEFICI

Combinazione tra la nozione di database e quella di interfaccia utente:

- link permettono di strutturare la conoscenza
- link permettono di accedere alla conoscenza
- link permettono di invocare programmi (applicazioni)

Integrazione di vari media

Accesso incrementale a differente livello di dettaglio gestibile in modo dinamico