Progettare l'interazione attraverso l'azione: le interfacce open source

Claudia Gianelli

1. Open source, pratiche di progettazione e prodotti

Lo sviluppo di software *free* e *open source* (da ora F/OSS, cioè "*free and open source software*")[1] sembra implicare una revisione radicale dei ruoli assegnati a utenti e progettisti, a partire da un'ampia de-centralizzazione – almeno apparente – dei processi di sviluppo, non solo dei codici informatici, i cosiddetti "sorgenti" (*source*)[2], ma anche dei software effettivamente realizzati a partire da questi codici e, dunque, in parte anche delle interfacce di questi prodotti. In questo modo sia la struttura del software (dal codice all'interfaccia) che il progetto che ne gestisce lo sviluppo (es. strutturazione delle comunità di programmatori e gestione delle pratiche che ne derivano) vengono ad essere caratterizzati da forme di modularizzazione distribuite su differenti livelli. In particolare, la modularizzazione riguarda non solo i due aspetti distinti del lavoro tecnico sul codice e la gestione del processo di sviluppo del software che ne deriva, ma anche l'articolazione fra di essi secondo veri e propri "progetti". In questo senso, la modularizzazione costituisce una condizione di possibilità proprio per l'articolarsi di questi vari livelli sino al rilascio di un prodotto finale agli utenti.

Il codice informatico, il cosiddetto "sorgente", è, in effetti, una sorta di unità minima del progetto, a partire e attorno al quale questo viene a configurarsi. L'efficacia di tali progetti pare allora derivare dalla capacità di prendere origine da questo codice e di crescere attraverso di esso. In questo senso il codice informatico si rivela non solo linguaggio di scambio per una sorta di comunità di adepti, che è in grado di manipolarne lo sviluppo e di guidarne l'evoluzione, ma anche oggetto di valore condiviso per una comunità più ampia. Attorno a questo lavoro propriamente tecnico vengono a costituirsi progetti articolati su più livelli e dimensioni, di cui possiamo seguire e studiare le linee di sviluppo fino ad arrivare a descrivere l'articolazione delle comunità di utenti e di sviluppatori e le loro dinamiche interne. Il codice informatico propriamente inteso diviene allora rilevante e interessante perché è questo codice che vincola e informa l'evoluzione del progetto che nasce per svilupparlo – orientando non solo gli sviluppatori che vi lavorano ma anche gli utenti che vengono coinvolti nella diffusione del prodotto finale.

Il codice sorgente stesso può quindi essere inteso già come corpo materiale, che diviene e che si forma, che viene dinamicamente ad avere senso perché manipolato, risultando continuamente agente e agito. Un codice che diviene a tutti gli effetti un attore, che interagisce con altri attori umani e non umani: questo è ciò che gli consente di circolare, di essere modificato, di essere distribuito.

E, ovviamente, di essere, infine, diffuso come un prodotto fruibile dagli utenti, e nella maggior parte dei casi, come un'interfaccia con cui gli utenti inter-agiscono. Particolare attenzione può essere in questo caso rivolta a come grandi comunità di sviluppo riescano a mantenere una gestione dinamica ed efficiente del progetto e, allo stesso tempo, a incrementare il numero dei partecipanti. Questo è quello che è stato definito come il passaggio dalle grandi cattedrali del software al bazar di Linux (Raymond 1998). Nel primo caso (e ancora nel caso di stretto free software come GNU) i progetti sono caratterizzati da una gerarchia interna, accettata dalla comunità, che gestisce strettamente l'accesso di nuovi elementi e competenze. In questo modo la programmazione del codice è connessa a una forma di socialità e di gerarchia interna, definita dal saper-fare dei vari partecipanti.

Parallelamente, l'affermarsi di orientamenti più "commerciali" nell'ambito del software libero ha portato a sviluppare una tendenza alla costruzione di comunità di supporto quanto più ampie possibili, sfruttando la rappresentazione pubblica e costantemente visibile della costruzione collettiva della comunità stessa. La stessa distinzione tra free ed open source sembra rivelarsi quanto mai complessa, più di quanto si sia discusso finora, così come la distinzione ideologica fra software "aperto" e proprietario. La dimensione sociale e comunitaria sembra allora richiedere nuove analisi del processo stesso di progettazione, che vadano oltre gli aspetti ritenuti meramente cognitivi e le abilità tecniche. Sembra in particolare necessario ridiscutere come si renda possibile una progressiva integrazione di contributi personali con la macro-dimensione del collettivo[3].

2. Il processo del design "aperto"

Da dove hanno origine progetti di questo tipo? Essa è solitamente associabile a una qualche necessità tecnica da parte di uno o più sviluppatori software: la necessità di migliorare una prestazione, un'azione o di creare una nuova soluzione per un problema non ancora considerato. Questo è l'avvio di una prima manipolazione

EIC Serie Speciale
Anno III, nn. 3/4, 2009, pp. 173-182

ISSN (on-line): 1970-7452
ISSN (print): 1973-2716

del codice, in qualche modo indipendente dal software nella sua totalità, e in ogni caso precedente ad ogni possibile modifica alla struttura di interfaccia, nonché talvolta totalmente slegata dall'idea stessa di interfaccia rivolta all'utente finale (a esempio nel caso dello sviluppo di software del tutto nuovi). A questo punto, la modifica potrebbe rimanere solo una soluzione locale, senza alcuna influenza sul progetto complessivo. Comunque, è del tutto comune che un bisogno "personale" riveli di essere collettivo e diffuso; il lavoro di alcuni singoli programmatori accentua o costruisce ex novo un bisogno, così che tutti (o la maggior parte) dei programmatori vengono a essere spinti a divenire parte di un'azione collettiva per soddisfarlo, al fine di rilasciare un codice sempre disponibile e modificabile.
Più ampia è l'azione, più forte è lo sviluppo di concrete procedure di risoluzione del problema: per ogni nuovo errore rivelato dal codice, il gruppo lavora per risolverlo. Questa serie di passaggi richiede ovviamente un tempo sufficientemente lungo per giungere ad un risultato ottimale, ciò implica una continua alternanza di ruoli: ad esempio un programmatore-utente rileva un *bug*, un altro programmatore ne valuta le possibili conseguenze, un altro *coder* ancora propone una possibile soluzione. Come si può vedere, questa struttura di risoluzione dei problemi potrebbe essere caotica o perfino impossibile da gestire: l'affermazione di procedure informali per la gestione delle modifiche previene ciò, poiché queste strutture informali divengono spesso assai "formali" nell'uso. Infatti, non è del tutto vero che chiunque possa cambiare il codice ovunque: le modifiche sono sempre in qualche modo approvate, attraverso gradi differenti di valutazione, da tutta la comunità o da un piccolo gruppo di leader di progetto. Nel primo caso, la valutazione collettiva è limitata a un giudizio pratico che si fonda su una dicotomia stretta fra funzionale e non funzionale. Il codice è messo alla prova al fine di comprenderne l'efficacia, il funzionamento, e le future implicazioni. Tuttavia, è nel secondo caso che il codice (e poi il software finale) viene messo alla prova e le modifiche o nuove funzioni sono valutate al fine di verificare se siano o meno integrabili nel progetto, distinguendo fra "ciò che è tecnicamente possibile" e "ciò che si deve fare". Il compito principale dei vari leader di progetto è quindi di mantenere un'immagine complessiva dello stato del prodotto, valutando la disponibilità di risorse e definendo – in modo più o meno formale – linee guida per la progettazione.
Per riassumere, abbiamo quindi almeno due differenti linee d'azione: la prima riguarda tutti i possibili sviluppatori-programmatori ed è connessa al livello più profondo del puro codice, dove i vincoli e le possibilità sono di natura prevalentemente tecnica, mentre il secondo riguarda solo alcuni sviluppatori-programmatori e tiene conto anche di tutta una serie di relazioni esterne – con gli utenti, le case produttrici dei software, gli altri software, ecc. – e di natura più generalmente strategica.
La distinzione fra queste due procedure di azione distinte ci consente di definire i due principali tipi di partecipanti dei progetti F/OSS: un vasto numero di utenti-programmatori e, dall'altra parte, un gruppo ristretto di programmatori/sviluppatori (veri e propri designer). Questi ultimi sono definiti in quanto tali perché hanno il ruolo di orientare e guidare il progetto al fine di mantenere una certa direzione e coerenza. Questa distinzione sembra rimarcarsi se guardiamo nuovamente alla distinzione fra free e open source: i secondi progetti hanno, infatti, un numero minore di utenti con competenze medio-alte nell'ambito dello sviluppo software. Questo costringe i progetti a diversificare le possibilità di partecipazione, anche per gli utenti con poca o nulla formazione tecnica. Si pensi ad esempio al gruppo degli utenti di Mozilla Firefox: troviamo un grande numero di utenti a far parte del gruppo di diffusione (*spread*), ma molti meno che lavorano direttamente sul codice.
In più, in quest'ambito non si sviluppano soltanto "prodotti" in vista di un certo uso, ma soprattutto si formano reti di collaborazione e nuove forme di partecipazione (Ducheneaut 2005). È in ragione della progettazione, implementazione e diffusione di un programma a partire da un codice sorgente aperto che vengono a svilupparsi strutture dense e ramificate che favoriscono la collaborazione fra diversi attori e a più livelli.
Per queste ragioni ci sembra che l'interesse per questo ambito da parte dei semiologi che si occupano di design possa essere per lo meno duplice: da un lato esso consente di osservare come si configurano le relazioni e le pratiche nel corso del processo di progettazione, dall'altro esso consente di analizzare quanto e come il prodotto finale della progettazione – cioè il software così come è utilizzabile dall'utente finale – abbia inscritte in sé tali relazioni e tali pratiche. Anzi si potrebbe dire che proprio nel passaggio da pratiche a prodotti e, quindi, nuovamente a pratiche, sembra trovarsi il punto di principale interesse per una semiotica che voglia occuparsi di studiare il F/OSS nei suoi vari aspetti. Ciò peraltro lo rende particolarmente interessante per indagare modalità, problemi e limiti del design dell'interazione.

3. Tracce di pratiche e analisi

Le pratiche proprie al design delle interfacce si inseriscono nel processo generale di progettazione in relazione a come viene sviluppato il codice sorgente, dato anche che l'interfaccia diviene livello di manifestazione di questo. Per quanto riguarda la definizione delle interfacce per i propri prodotti, il software "aperto" sembra ancora dividersi fra due tendenze opposte: da una parte non sembra ancora essersi reso autonomo rispetto ai modelli dei propri omologhi proprietari, dall'altra mostra comunque alcune interessanti linee di sviluppo e di innovazione (Nichols-Twidale 2003). La questione può rivelarsi esemplare rispetto al problema dell'innovazione delle interfacce se pensiamo a quest'ultima non come alla revisione di singoli elementi ma, ad un livello

più generale di organizzazione e gestione dell'azione e dell'interazione: in questo senso innovare vuol dire lavorare radicalmente sulla relazione fra elementi differenti – si tratta di interazione a vari livelli, non solo fra un'interfaccia data e un utente tipo.
Come si diceva poc'anzi, il F/OSS non si presenta mai esclusivamente come prodotto da analizzare, ma permette sia direttamente sia indirettamente, di interrogare le pratiche di progettazione. In particolare, come si è visto, la struttura dei progetti OS, e dunque di tutto quello che sta "prima" del prodotto, è assai accessibile e trasparente e ci consente dunque di metterla direttamente in relazione con ciò che possiamo analizzare di quel prodotto, ad esempio l'interfaccia finale con cui gli utenti andranno a interagire. La particolarità delle interfacce dei prodotti "open" sembra dunque risiedere nel fatto che esse sono oggetti che chiaramente mantengono traccia, in alcuni casi assai visibile, delle pratiche che le hanno prodotte. In questo senso, tali tracce consentono, attraverso la descrizione del loro sviluppo e del loro divenire dinamico, di avere accesso alle pratiche stesse che le hanno prodotte. Nel caso di alcuni prodotti le cui versioni vengono aggiornate di frequente, il F/OSS consente di avere accesso alle pratiche che letteralmente stanno producendo le tracce. Ciò è di grande rilevanza da un punto di vista metodologico, in quanto consente di articolare in modo assai complesso la relazione fra le varie pratiche e i prodotti risultanti da esse. Ciò consente inoltre di verificare che la descrizione delle une come degli altri sia duplice: da una parte non possiamo descrivere pratiche senza tener conto di come queste effettivamente siano caratterizzate da un qualche risultato, dall'altra non possiamo descrivere oggetti, artefatti materiali o interfacce "virtuali" scindendoli dalle pratiche che li hanno prodotti e da altre in cui sono continuamente presi. Da questo punto di vista non possiamo che vedere una continuità fra pratiche e loro prodotti che non potrà che riflettersi in una continuità del punto di vista metodologico per quanto riguarda la descrizione e l'analisi. Emerge, dunque, come gli oggetti – questi oggetti in particolare, cioè i prodotti F/OSS – siano un caso esemplare di come in semiotica possiamo avere accesso alle pratiche in modo specifico per la disciplina.
In generale, si può dunque affermare che è sempre possibile ricostruire l'immanenza di una pratica, cioè il diagramma immanente a una data pratica che rende conto di come essa si origina e si sviluppa. Tale diagramma non è per forza qualcosa di astratto, ma può trovarsi inscritto in un dato artefatto, come ad esempio capita nel caso delle pratiche di interazione rispetto alle quali esso si materializza nell'interfaccia che rende possibile quella stessa interazione.

4. Interfacce e F/OSS

4.1. Una definizione di interfaccia

L'analisi delle interfacce dal punto di vista specifico delle azioni che esse rendono possibili e che vi si realizzano, richiede che si definisca preliminarmente cosa si intenda per interfaccia e come questo influisca sulla particolare prospettiva adottata per studiare il design e i suoi prodotti. Un buon punto di partenza sembra essere quello della definizione data da Brenda Laurel e S. Joy Mountford (1990, p. XIII, trad. mia): l'interfaccia viene qui ad essere considerata come "superficie di contatto" (*contact surface*) che "riflette le proprietà fisiche degli interagenti, le funzioni che devono essere messe in atto e il bilanciamento tra potere e controllo". Sappiamo però bene che l'interfaccia non è solo "superficie", involucro, pelle, ma essa ha anche una struttura interna, una carne localmente determinata tanto da relazioni interne (relazioni strutturali propriamente dette, intese come "condizioni" e "vincoli", ad esempio) e da relazioni esterne (realizzate nell'uso). La progettazione contribuisce a costruire queste relazioni e a gestirle. Queste relazioni coinvolgono perlomeno due livelli distinti: da una parte quello che riguarda l'interfaccia come "spazio" o come superficie di azione, dall'altra quello che concerne l'interfaccia come spazio di mediazione tra le azioni svolte su essa (e come queste azioni vengono tradotte per altre istanze in relazione con l'interfaccia). In questo modo, l'interfaccia assume un ruolo di mediazione fra spazi e corpi, al fine di assicurare le connessioni fra di essi e in più fornire un orientamento (una "direzione") alle relazioni dinamiche che essi possono articolare.
In altre parole, possiamo dire che l'interfaccia non costituisce solo un involucro per una struttura interna, agendo per contenerla e per mediare con l'esterno, ma essa stessa può essere a sua volta articolata in termini di involucro e struttura interna e può, quindi, essere considerata a tutti gli effetti un corpo nei termini di Fontanille (2004). Un corpo, dunque, in relazione con altri corpi; non solo con i corpi degli utenti, ma anche con il corpo del codice informatico, considerato nella sua propria materialità.
Ogni interfaccia non è quindi un dato, ma è in continua costruzione e ri-costruzione a partire da procedure di installazione di attori sia umani che non umani, nonché a partire da movimenti di punti di vista. Un interfaccia non rileva, dunque, primariamente, o comunque non solamente, di procedure di visualizzazione. L'innovazione di un'interfaccia riguarda l'istituzione di nuove relazioni o la ri-mediazione di quelle già presenti, modificandone il modo di esistenza semiotica. L'interfaccia consente, ad esempio, di rendere *virtuali* alcune relazioni attuali, mentre altre azioni *potenziali* vengono attualizzate proprio grazie al suo ruolo mediatore. Si tratta quindi di costruire l'interfaccia non solo come uno spazio che contiene oggetti, ma soprattutto di articolarla come una superficie dotata di una propria densità relazionale, che venga a rendere questo spazio a tutti gli effetti "praticabile".
Sarà poi necessario prendere in qualche modo le distanze dagli studi che tendono a vedere le interfacce come

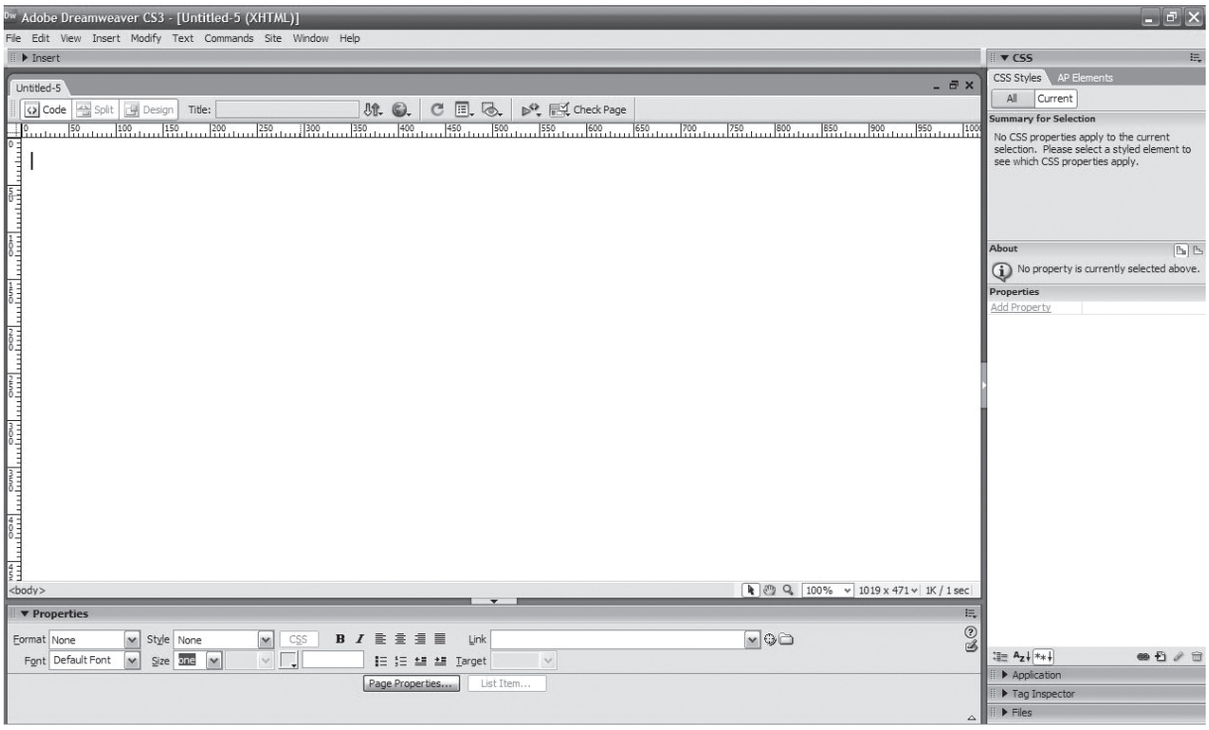

Fig. 1 – Interfaccia base di Dreamweaver CS3

un problema di natura eminentemente cognitiva, in relazione all'esecuzione di determinati compiti e alla gestione di risorse. Innanzitutto, per non opporre radicalmente cognizione ed azione, così come azione e passione. Come d'altra parte gran parte degli studi sulla cosiddetta *embodied cognition* sembrano assumere, non vi è che continuità fra cognizione, percezione ed azione. È bene rimarcare che la semiotica ha già rilevato questa continuità e può efficacemente metterla in gioco in sede di analisi. In questo senso, la cognizione non è che uno dei modi possibili di darsi della semiosi, uno dei possibili modi di produzione del senso.

4.2. Livelli di analisi delle interfacce

Per analizzare le interfacce, in primo luogo occorre considerare un livello eminentemente plastico, dove l'interfaccia è rilevabile come una specifica configurazione di relazioni tra forme e colori. A partire da questa configurazione plastica l'interfaccia può emergere a sua volta come un vero e proprio corpo, attraverso l'affermarsi di una struttura interna e di un involucro propri. L'interfaccia in quanto corpo entra così – come accade per gli oggetti (Mattozzi 2009) – in relazione con altri corpi, siano essi umani, dotati ad esempio di una specifica gestualità, siano essi non umani. Da qui emerge a sua volta l'ultimo livello in cui l'interfaccia viene ad entrare in relazioni più complesse con altri oggetti e soprattutto con pratiche sia individuali che sociali e socializzate.

L'articolazione degli spazi viene così a determinare la forma dell'azione generale e dei singoli atti e movimenti. In questo modo una specifica articolazione di spazi diviene cruciale in quanto determina un vero e proprio ritmo dell'azione e dell'interazione, vincolando dunque le pratiche che verranno effettivamente a realizzarsi. Possiamo così rilevare la possibilità di dare dell'interfaccia una descrizione che sia propriamente "formale", intendendo con questo l'esplicitazione delle condizioni di possibilità per la realizzazione di determinate pratiche, così come della virtualizzazione di altre.

Per applicare queste categorie generali – come spazio, relazione, corpo – abbiamo scelto di lavorare sull'analisi di due software (oggetti?) abbastanza simili nei loro rispettivi obiettivi di progettazione che presentano entrambi le caratteristiche WYSWYG ("what you see is what you get"). Si tratta di Dreamweaver CS3 di Adobe e NVU 1.0, un software sviluppato in ambito open source all'interno della cosiddetta Mozilla Foundation. Entrambi i programmi consentono la progettazione e l'editing di pagine web e di interi siti, a partire da strumenti di composizione grafica e di gestione dei contenuti.

Questi strumenti di creazione e impaginazione sono simili nello stabilire un set specifico di azioni tipiche in termini di procedure gestuali e scelte ristrette. Un limitato repertorio di azioni viene definito per consentire agli utenti di pre-vedere (nel senso di prevedere effetti, ma anche di vedere in anteprima) ogni risultato delle proprie azioni. Essi si trovano così a poter manipolare effetti grafici e quant'altro, senza dover accedere direttamente ai codici di base che generano le pagine web, come il codice HTML o la strutturazione in CSS. In altre parole, ciò consente agli utenti di manipolare direttamente e in tempo reale il risultato finale della propria

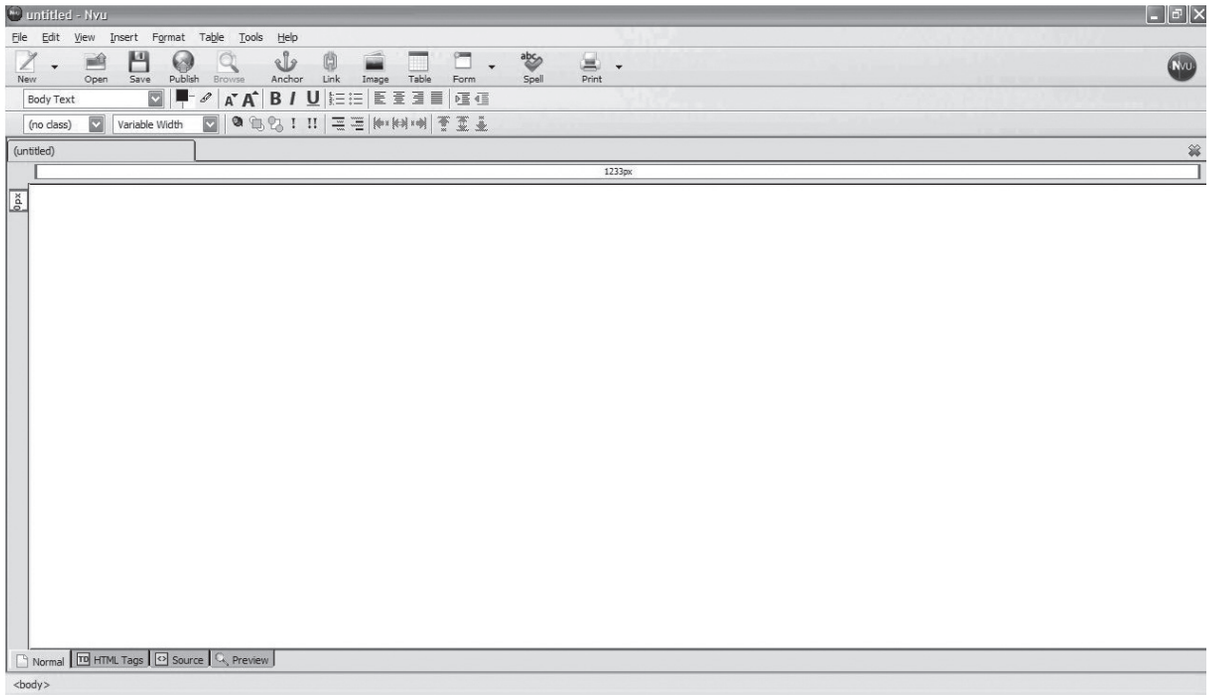

Fig. 2 – Interfaccia di NVU

attività di progettazione, cioè di lavorare concretamente sugli effetti delle proprie azioni. Allo stesso tempo è il programma stesso che genera un codice HTML a partire dalle scelte degli utenti, mantenendo però la possibilità per l'utente esperto di apportare modifiche direttamente a questo livello qualora si riveli necessario. Le interfacce di tali prodotti si rivelano quindi assai complesse, dal momento che è a partire da esse che si determinano le possibilità di azione per vari utenti e, come si dice, la loro "soddisfazione".

Questi programmi condividono quindi la realizzazione di un medesimo programma narrativo: non solo progettare e disegnare pagine web, ma anche vederle e verificarle costantemente, lavorando sul momento e sugli effetti delle proprie azioni. La possibilità che viene data agli utenti è di lavorare al contempo sia (eventualmente) sul codice HTML della pagina web in corso di creazione – che sta a un livello inferiore – sia sulla manifestazione, ovvero ciò che il browser mostra, e di essere costantemente coinvolti nel processo di generazione dell'interfaccia del proprio sito.

L'organizzazione dello spazio nei due programmi è orientata radicalmente all'azione, alle possibilità di lavorare su codice HTML, stili, elementi grafici e contenuto. Dunque, se possiamo considerare i risultati finali sostanzialmente equivalenti, tuttavia dobbiamo anche supporre che sia invece in atto una differenziazione fra i due software a livello dei task che devono essere compiuti al fine di manipolare oggetti come codice HTML o layout grafico.

Consideriamo i *task*[4], le "operazioni" o i "compiti" come comprendenti sia le sequenze possibili di azioni e gesti, sia gli oggetti verso cui queste azioni sono dirette. In questo modo possiamo renderne conto non come unità predefinite o comportamenti stereotipici, ma come la realizzazione di configurazioni specifiche, localmente definite dalla relazione fra interfaccia ed utenti.

L'analisi delle interfacce deve dunque tenere conto di almeno due possibili livelli: da una parte NVU e Dreamweaver in quanto software completi che consentono di generare pagine web dotate di una loro interfaccia; dall'altra i due software considerati in quanto "prodotti" di un processo di design che ha portato alla realizzazione di una determinata interfaccia-utente che consente di gestire in modo più o meno rigoroso le azioni. In terzo luogo, e qui si trova la novità che intendiamo studiare, NVU in quanto prodotto "open" si caratterizza per un particolare sviluppo non solo del proprio codice sorgente, ma anche della propria interfaccia-utente, a cui si può aver accesso nel momento in cui si prendono in considerazione le sue procedure di sviluppo. In particolare, terremo conto del fatto che l'interfaccia di NVU ci consente di ricostruire la sua formazione, avendo in qualche modo accesso alle pratiche che l'hanno generata, e che l'hanno generata tenendo conto anche delle successive pratiche in cui il software sarebbe stato coinvolto una volta terminato.

Infine, è necessario considerare che NVU, così come altri software, non nascono nel vuoto, ma all'interno di un *milieu* in cui già esistono software simili, con cui non possono non entrare in relazione. In particolare, nel caso di NVU non si può non considerare che il processo di progettazione non ha potuto non tener conto dell'esistenza di Dreamweaver, assumendolo come base, per poter poi ricostruirne e ricombinarne la struttura.

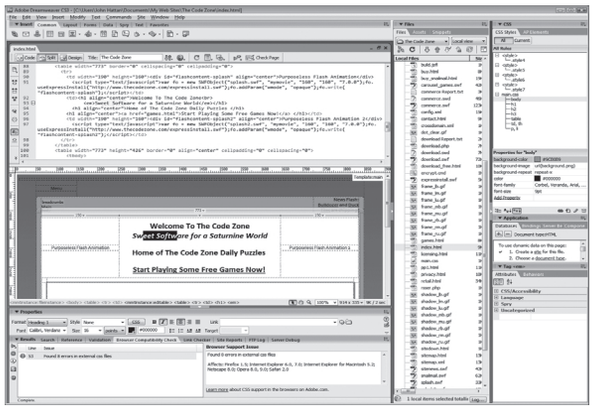

Fig. 3 – Visualizzazione "split" in Dreamweaver

4.3. Costruzione di spazi enunciazionali e pratiche

Considereremo lo spazio complessivo dell'interfaccia come l'articolazione di sotto-spazi o, meglio, come l'articolazione tra diverse enunciazioni (con i loro rispettivi enunciati) manifestate da cornici che delimitano uno spazio specifico (tradizionalmente quelli che definiamo "finestre", ma che non sono le uniche enunciazioni, dato che tra queste possiamo contare anche barre, tasti e zone di intervento)[5]. L'interfaccia è dunque costituita da quelli che potremmo chiamare "spazi enunciazionali", indicati nel seguito dell'articolo semplicemente come "spazi". L'articolazione tra questi spazi può essere statica o dinamica. Nel primo caso l'articolazione è manifestata dal layout attraverso varie possibilità di incassamento (una finestra dentro l'altra; come ad esempio nella fig. 2, in cui la finestra di visualizzazione "Normal" è incassata nella finestra più generale dell'interfaccia di NVU) o giustapposizione (una finestra affianca l'altra, come ad esempio nella fig. 1, dove la finestra di visualizzazione di Dreamweaver è giustapposta alla finestra verticale dei CSS). Nel secondo essa è messa in atto nella pratica e si sviluppa come la creazione di una concatenazione tra vari spazi che realizza una traduzione tra uno spazio e l'altro (l'esempio più classico è il passaggio dalla una visualizzazione ad un'altra da *Code* a *Design* in Dreamweaver o da *Normal* a *Source* in NVU"[6]).

Se consideriamo gli spazi dal punto di vista delle pratiche potremo genericamente rilevare:

- pratiche di articolazione dello spazio dell'interfaccia che si dispiegano nel momento della progettazione – ad esempio la decisione presa dal designer di giustapporre o incassare due finestre (sia nell'interfaccia del software – NVU o Dreamweaver – sia nella pagina web creata con questi software);
- pratiche presupposte dall'interfaccia e dalla disposizione dei suoi spazi che consentono di modificarli e modificarne le relazioni – come si vedrà in seguito, Dreamweaver dispone di una visualizzazione giustapposta (*Split*) (fig. 3) alternativa alle visualizzazioni *Code* e *Design* cosa che NVU non ha;
- pratiche di uso effettivo dell'interfaccia che ne articolano i vari spazi – queste sono effettivamente quelle a cui mi riferivo prima quando parlavo di concatenazioni di spazi – in NVU, ad esempio, si potrà dispiegare un specifica alternanza tra visualizzazione *Normal*, *Source* e *Preview*.

Ciò che distingue ciascuna di queste pratiche non è la loro supposta maggior o minor "realtà" – per cui quelle presupposte sarebbero meno "reali" di quelle effettive – quanto la loro diversa densità relazionale. L'efficacia di un'interfaccia sarà allora valutabile solo rispetto alla densità delle relazioni da essa articolate, siano esse predisposte (virtuali) o effettuate (realizzate). Tale riflessione può essere estesa anche esternamente all'ambito delle interfacce e, dunque, lo spazio di un'interfaccia è relazionalmente denso in modo differente dallo spazio che considereremmo come "reale". Ciò non vuol però dire che vi sia una differenza di esistenza fra le varie pratiche messe in atto in spazi "reali" o in spazi "virtuali"– per cui alcune sarebbero più "reali" di altre –, perché ogni pratica che si compie per mezzo e su un'interfaccia è tanto "reale" quanto quelle che compiamo su oggetti di cui è chiaramente rilevabile una materialità.

La costituzione di questi differenti spazi, che sono sempre spazi per l'azione, rende possibile, in relazione alle pratiche che a partire da essi si possono dispiegare, una selezione delle condizioni di visibilità e del modo di esistenza di alcuni aspetti di ciò che, attraverso l'interfaccia di un software come NVU o Dreamweaver, si viene a creare, cioè la pagina web o il sito con la propria specifica interfaccia. Da ciò risulta che l'interfaccia del software non solo orienta la pratica di progettazione di una pagina o sito web, ma anche il suo stesso risultato.

Se, dunque, teniamo conto della concatenazione di relazioni che si pongono tra una data pratica, il risultato di questa e come quest'ultimo predispone un'altra serie di pratiche che si vengono a loro volta a dispiegare, producendo altri prodotti e conseguentemente altre pratiche, possiamo allora ipotizzare che un processo di design che si appoggia a un dato software possa trasporre nel prodotto di tale processo le caratteristiche del software e delle stesse pratiche che lo hanno generato. Prodotti sviluppati tramite software OS mostreranno dunque le tracce delle pratiche OS.

Cercheremo di dimostrare ciò a partire dall'analisi comparativa delle interfacce di NVU e Dreamweaver, che permetterà di mettere in luce non solo le pratiche che da esse si dispiegano, ma anche quelle che presuppongono.

4.4. Configurazione dell'interfaccia: dal codice sorgente al progetto

A fronte di un programma narrativo che è il medesimo – creare pagine e siti web –, dobbiamo considerare come esso viene differentemente articolato e modulato da parte dei due programmi, soprattutto per quanto riguarda il livello del codice HTML. Ciò risulta alquanto

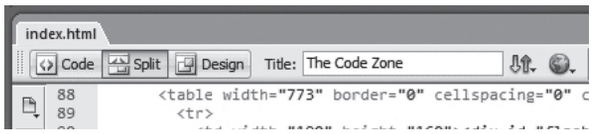

Fig. 4 – Uso delle tab e posizionamento dei punti di intervento in Dreamweaver

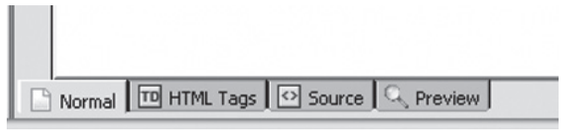

Fig. 5 – Uso delle tab e posizionamento dei punti di intervento in NVU

evidente se compiamo anche solo una veloce analisi lessicale delle due etichette (*label*) utilizzate dall'interfaccia per identificare le stesse sequenze di codice HTML o XML: NVU lo chiama "*source*", sorgente, come l'origine e il vero inizio dell'intero processo di progettazione, facendo al contempo riferimento al lessico OS. Dreamweaver, invece, sceglie un termine ben più neutrale, cioè "codice", considerandolo in modo più tecnologico, come il prodotto di una azione di un attore specifico, di un programmatore, un *coder* appunto.
Se si osserva la finestra principale (figg. 1 e 2), si può notare che le varie possibilità di visualizzazione sono strutturate in modo diverso nei due software (figg. 4 e 5). La prima cosa che salta agli occhi è che in NVU ci sono quattro possibili visualizzazioni di ciò che si sta creando – *normal*, *HTML tags*, *source*, *preview* – mentre in Dreamweaver sono possibili, a partire dagli elementi posti sul margine della finestra, solo due tipi di visualizzazione – *Code* e *Design*, che possono anche essere affiancate nella visualizzazione *Split*.
Se queste differenti possibilità di visualizzazione, con le specifiche etichette utilizzate per definirle, strutturano in modo diverso l'interazione con i due software, ciò su cui vorrei soffermarmi è qualcosa di probabilmente meno visibile, ma a mio parere non meno rilevante. Si tratta del posizionamento delle etichette. Non tanto il contrasto di carattere topologico – in NVU sono poste in basso, mentre in Dreamweaver sono poste in alto –, quanto il posizionamento in termini enunciazionali. In NVU, i quattro punti di intervento, che hanno una configurazione simile alle *tab*[7] di scelta delle pagine di lavoro, sono parte della cornice, mentre in Dreamweaver sono inseriti all'interno della cornice, come tasti. NVU, dunque, delinea ciascuna modalità di visualizzazione come fosse una scheda autonoma che fa riferimento ad una certa pagina, mentre in Dreamweaver sono solo diverse visualizzazioni del contenuto della stessa cornice. NVU in questo modo valorizza l'autonomia di ciascuna visualizzazione, che può anche non coincidere con un altro tipo di visualizzazione; tant'è che tra le scelte prevede anche la *preview* che non è automaticamente determinata dal codice soggiacente, ma dall'interazione tra questo, il browser e spesso una serie di plug-in di cui il browser dovrebbe essere provvisto. Diversamente, Dreamweaver, che pone il punto di intervento per l'azionamento della *preview* in altro spazio, presenta le due modalità di visualizzazione come versioni diverse l'una dell'altra, valorizzando così la loro interscambiabilità. Dreamweaver è più assertivo, ci dice: "dentro quella finestra trovi la stessa cosa, anche se visualizzata differentemente". NVU è più cauto: "ogni pagina ha una sua autonomia, anche se sono interrelate, verifica sempre le corrispondenze, magari facendo una *preview*". Ecco che allora l'uso del termine "sorgente" (*source*), rispetto al più neutro "codice" (*code*) ha una sua rilevanza, dato che, pur nella loro autonomia e parità, un tipo di visualizzazione – *source*, per l'appunto – viene indicato come più fondamentale degli altri.
Questa strutturazione gerarchica delle visualizzazioni viene a riprodursi nell'organizzazione spaziale delle cosiddette *tab* in cui le visualizzazioni stesse vengono ad essere suddivise durante il lavoro.
Le relazioni fra questi elementi possono, infatti, essere considerate dal punto di vista dei loro rispettivi modi di esistenza semiotica. Ognuno di essi può essere considerato come la realizzazione di un altro, o ancora come la virtualizzazione di alcune proprietà. Per esempio, in NVU la *preview* può essere assunta come la realizzazione/manifestazione di tutti gli altri livelli; o i *tag* HTML possono essere considerati sia come proprietà virtuali, realizzate poi nella visione normale, sia come l'attualizzazione intermedia del codice.
D'altra parte, in Dreamweaver, si nota che le relazioni fra *code/split/design* sono in qualche modo neutralizzate, tenendo anche conto del fatto che non è presente alcuna forma di *preview* nei comandi di base. Al contrario, è fortemente accentuata la sequenza logica di un mero processo di presupposizione di azioni: il codice HTML dovrebbe precedere la manifestazione (il design grafico) sia da un punto di vista logico che funzionale[8].

4.5. Pratiche e spazio di lavoro

Ma Dreamweaver dispone anche un altro tipo di visualizzazione che, pur mantenendo una corenza con quanto visto, permette di ri-mediare le azioni in base alla configurazione dello spazio di lavoro. Dreamweaver infatti presenta due modalità di strutturazione dello spazio di lavoro: Coder View e Designer View. In questo caso gli spazi vengono ri-organizzati e ri-combinati in funzione del tipo di relazione che l'utente può stabilire con il prodotto. Questo comporta uno slittamento nelle azioni presupposte e attese nonché una modificazione rispetto all'oggetto che viene a costruirsi. Per quanto riguarda il *coder* (il programmatore), la colonna del menu principale (CSS e simili) è sulla sinistra: significa che le azioni qui previste sono logicamente presupposte all'azione di modificazione effettiva del codice. Questo implica una sorta di "puntualità" delle scelte, intesa in senso aspettuale. Al contrario, la stessa colonna passa

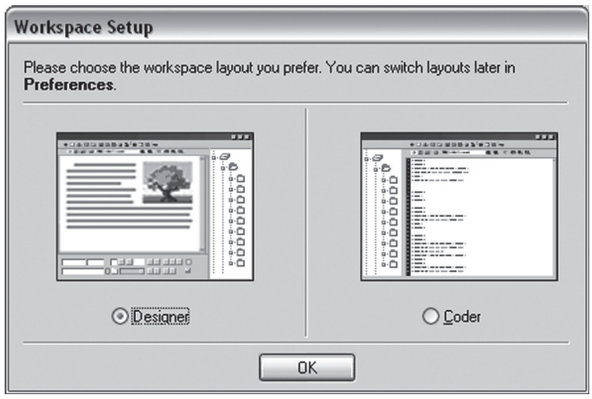

Fig. 7 – In Dreamweaver la possibilitò di selezionare tra *Designer* e *Coder*

sul lato destro quando il layout nella versione designer è attivo. Questo implica che le scelte principali vengano ordinate al fine di seguire la creazione del layout grafico per la pagina web. Dunque, in questo caso, gesti ed azioni sono supposti essere iterativi e ripetuti in questo caso.

Da quanto abbiamo visto, nelle due applicazioni si prefigura una differente distribuzione delle sequenze di azioni che vanno a costituire *task* autonomi – vale a dire programmi narrativi come quello di creare un gruppo di pagine collegate – con un orientamento che può essere sia al processo sia al risultato finale. Nel primo caso verrà data rilevanza all'organizzazione delle azioni, al "come" si opera per raggiungere determinati risultati (es. costruire una singola pagina), e dunque alle relazioni di tipo gerarchico o di presupposizione che vengono a stabilirsi fra i vari passaggi della progettazione. Nel secondo, invece, verrà posto l'accento sul risultato finale, il quale può essere raggiunto attraverso una molteplicità di azioni alternative, che dipendono da necessità e adattamenti locali. Le due prospettive, ovviamente, non si escludono a vicenda, ma anzi possono convivere, e la capacità di coniugare i due orientamenti è quello che rende un programma efficiente ed efficace nella relazione con diversi utenti, siano essi esperti (designer grafici o programmatori) o principianti.

Nel caso di Dreamweaver le due visualizzazioni *coder* e *designer* ben rispecchiano la presenza di un orientamento al risultato, contribuendo a costruire un effetto di professionalità. Strutturando gli spazi di lavoro attraverso due visualizzazioni, una per il programmatore esperto (*coder view*), l'altro per il designer grafico (*designer view*), viene a costruirsi uno spazio generale di lavoro dove le possibilità di azione per l'utente esperto vengono a moltiplicarsi. Viene così a definirsi una cornice che è orientata dal risultato finale (in un caso un codice ottimizzato, nell'altro il layout grafico) all'interno della quale l'esperto può muoversi e organizzare autonomamente le proprie sequenze di azione, in base anche alle proprie pratiche "tipiche" e alla propria esperienza. Tuttavia, l'articolazione di due spazi diversi e pre-definiti fa sì che le pratiche vengano considerate sì come strettamente dipendenti dall'esperienza, ma anche come definibili a priori da un esperto.

Nel caso di NVU la relazione con le pratiche di utilizzo viene a configurarsi in modo differente, anche in base all'organizzazione degli spazi di lavoro. In questo caso sembrano convivere e articolarsi in modo più complesso i due orientamenti al processo e al risultato, in una sorta di "apertura" della struttura in cui le azioni possono prodursi. In questo senso NVU sembra tener conto in modo più efficace di come le pratiche di programmazione e design che esso rende possibile sono tutt'altro che logiche, lineari, strutturate, ma al contrario in continuo adattamento locale e ri-combinate a seconda delle esigenze e competenze dei diversi utenti. Ad esempio, se è possibile individuare delle sequenze tipiche di azione, queste possono sempre – entro certi limiti – essere ri-combinate e riprodotte con un certo grado di differenziazione.

Diversamente la cornice dello spazio di lavoro di NVU viene mantenuta attraverso tutti i livelli possibili di visualizzazione meno uno, quella "*source*". In tutti gli altri la configurazione plastica si stabilizza attraverso il mantenimento di alcuni elementi che compongono lo spazio di lavoro. In particolare, attraverso il mantenimento delle relazioni fra questi elementi (per esempio il posizionamento reciproco dei punti di intervento) che sono parte integrante delle pratiche di progettazione che vengono a svilupparsi in questo spazio. In questo modo, le pratiche vengono "guidate" ma non vincolate alla presenza di questi elementi secondo una precisa disposizione.

Ciò, tuttavia, viene a cadere nel caso dell'unico livello di visualizzazione in cui la configurazione si modifica e la continuità viene a rompersi. Questo è il caso della visualizzazione *source* (fig. 8): lo spazio viene ad aprirsi, così come aperto è il codice. Lo spazio si costituisce come meno vincolato e vincolante, operando per mediare pratiche che vengono considerate meno fisse e tipiche, caratterizzate da una sorta di creatività che questo stesso spazio "libera". Spazio aperto che si dispone alla relazione con utenti esperti, dotati di una propria gestualità, di un ritmo e di un'esperienza di pratiche che rileva della loro competenza: essi sanno cosa fare, ma soprattutto sanno come farlo al meglio e come muoversi nello spazio che viene fornito ri-semantizzandolo all'occorrenza. Utenti che non necessitano di essere "guidati", poiché possiedono una visione più complessiva e organica del proprio lavoro, conoscendo, ad esempio, l'effetto di una modifica sul codice sulla manifestazione grafica e viceversa, e potendo dunque orientarsi da subito verso un risultato atteso o previsto. Lo spazio risulta così, allo stesso tempo, costante e frammentato, suddiviso per sostenere relazioni differenti e orientamenti mutevoli, ora al processo e alle singole azioni, ora al risultato finale. Questo è evidente nelle visualizzazioni "*normal*" e "*html tag*", che rispetto alla visualizzazione

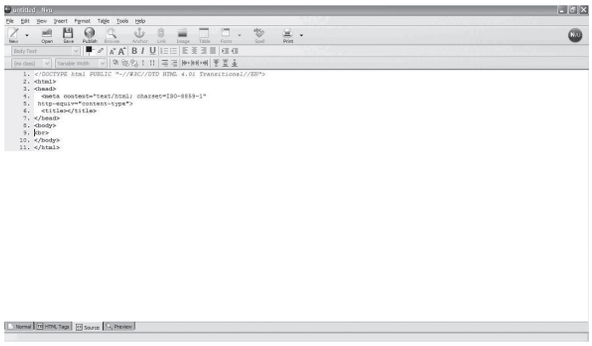

Fig. 8 – Visualizzazione source in NVU

code di Dreamweaver presuppongono utenti vincolati a specifiche sequenze di azione organizzate al fine di ottenere singoli risultati provvisori e un risultato complessivo soddisfacente. In questo senso, la realizzazione corretta di ogni singolo passaggio (programa narrativo d'uso, dunque) è condizione necessaria perché il processo possa proseguire ed arrivare a compimento.

4.6. Pratiche e standard: browser e *tab*

L'ultimo passaggio di cui NVU e Dreamweaver si occupano è quello che riguarda l'effettiva visualizzazione e utilizzo della pagine web che i due software consentono di produrre. Si tratta del passaggio cruciale alle pratiche di navigazione effettivamente realizzate, che devono tener conto dei vincoli strutturali forniti dai browser e di come la prassi si adegua a questi vincoli.

Il browser assume un fondamentale ruolo di sanzionatore, operando per regolare lo spazio e il tempo dell'interazione, risultando così non solo uno strumento, ma anche un attore di mediazione fra designer, pagine web, siti e utenti. In particolare, i vincoli strutturali dati dal browser gestiscono le possibilità di navigazione all'interno di una pagina web e fra pagine e siti differenti. Questo costringe in qualche modo ad adattare gli altri elementi della progettazione agli standard di navigazione definiti da specifici software. L'influenza non è però necessariamente unidirezionale, dal browser alla navigazione, ma riguarda anche come determinate pratiche di navigazione finiscano col modificare la struttura e gli standard dei browser stessi.

Un esempio particolarmente rilevante in questo senso riguarda il ruolo dell'introduzione delle cosiddette *tab* (o "schede di navigazione") nell'affermazione del browser Firefox – prodotto OS sviluppato da Mozilla Foundation. Ne è conferma il fatto che tale funzione oggi è stata estesa non solo ad altri browser (come Explorer dalla versione 7), ma anche – come si è già visto – a programmi di altro tipo.

Queste schede nella loro funzione originaria consentono di aprire e visitare un grande numero di pagine web all'interno di una stessa finestra. Questo rende più agile e veloce il passaggio da una pagina all'altra, riducendo (collassando) il tempo e lo spazio di interazione. Non solo, si assiste ad una vera ri-combinazione di spazio e tempo, dal momento che essi divengono qualcosa di differente da semplici vincoli: contribuiscono a definire localmente l'azione, in termini di differenziazione delle possibilità di azione che vengono così a moltiplicarsi e consentono di rompere gli schemi di azione stereotipici a cui di solito ricerca, navigazione e lettura vengono ricondotti.

Concretamente, poi, questa funzione non si limita a riorganizzare la gestione degli spazi di interazione, ma va a interagire con altri fenomeni che hanno di recente contribuito a modificare i cosiddetti "stili di navigazione". Ci riferiamo, a esempio, a nuove forme di produzione e pubblicazione di contenuti on-line: si pensi ad esempio ai cosiddetti *weblog*. La forma generale è quella di gestione di contenuto in aggiornamenti relativamente frequenti, orientata al dare immediata visibilità al nuovo contenuto e ad ogni forma di aggiornamento.

Questa possibilità di economizzare anche la lettura e la consultazione dipende dalla configurazione spaziale: infatti, una struttura disposta su colonne uniche è emersa come efficace ed efficiente proprio per la facilità di gestire contenuti e per la capacità di rendere immediatamente percepibile e visibile lo spazio più rilevante, cioè quello dei *post* dove gli aggiornamenti sono più frequenti. Da questo punto di vista è possibile sia percepire una continuità data dal mantenimento di una configurazione plastica (di colori, linee, forme) sia la rottura data dall'aggiornamento. Un blog è quindi una forma discorsiva relativamente stabile in ragione del fatto che la sua struttura può essere "rotta" e allo stesso tempo "ristabilita": è anche questo che costruisce nel lettore un senso di attesa.

L'introduzione delle *tab* può essere forse dovuta proprio all'emergere di queste nuove pratiche di lettura, mostrandoci come gli standard e le pratiche tendano a migrare e possano essere tradotte da un ambiente all'altro, dalla pratica agli strumenti di navigazione. Questo sembra mostrarci una possibilità di affermazione di standard di tipo bottom-up, dal basso verso l'alto, consentito dal fatto che tutte le differenti pratiche che abbiamo considerato – di navigazione, sviluppo software, blogging, lettura – si sono sviluppate in una simile dimensione socio-culturale, caratterizzata da una generale "apertura".

5. Conclusioni

Come abbiamo brevemente mostrato, le pratiche realizzate assumono un ruolo cruciale nel design di interfaccia. In particolare, questo è coinvolto nella migrazione e definizione di alcune configurazioni considerate solitamente come "standard" o "tipiche". Alcuni progetti aperti ci hanno dimostrato che c'è la possibilità di lavorare su queste basi, ma cercando di rinnovare relazioni ed azioni. Lo studio di queste esperienze potrebbe consentire ai designer di comprendere come mantenere alcuni standard e allo stesso tempo dare origine a nuove e migliori forme di interazione. La nostra analisi com-

parata ci ha inoltre consentito di discutere come il design può operare su spazio ed azione al fine di orientare la realizzazione di specifiche pratiche.

Note

[1] Ci riferiamo a quegli ambiti di sviluppo software dove il cosiddetto "codice sorgente" (source), su cui gli sviluppatori lavorano per elaborare prodotti differenti – da semplici applicazioni a interi sistemi operativi – è un codice "aperto" ed accessibile. Questo in contrapposizione alle modalità di sviluppo delle grandi case produttrici (come Microsoft) che di solito mantengono i propri codici sorgente "chiusi" e non accessibili a modifiche. La sigla F/OSS include progetti di sviluppo molto differenti fra loro, non tanto per le modalità di lavoro che sono simili (decentrate, distribuite) quanto per le implicazioni più ideologiche del loro lavoro. La parte Free è quella più radicale, legata allo sviluppo di progetti come Linux, mentre la parte Open è più orientata all'effettivo avanzamento tecnologico dato da modalità di lavoro collettivo e di totale apertura, senza disdegnarne le implicazioni commerciali. Non è tuttavia facile (e spesso utile) operare questo tipo di distinzione.

[2] S'intende come "sorgente" o "codice" l'insieme di istruzioni in un determinato linguaggio di programmazione che consentono la creazione e lo sviluppo di un programma per computer. È la sequenza che fisicamente viene costruita da un programmatore al fine di "dialogare" con il computer e di far eseguire ad esso determinate azioni o sequenze di azioni complesse. L'apertura del codice è, quindi, cruciale per consentire l'evoluzione e il miglioramento del codice stesso.

[3] Si veda sugli aspetti anche materiali di questa integrazione Ducheneaut (2005).

[4] Faccio notare che il termine è utilizzato anche in studi di design, ergonomia, usabilità, anche se con un significato parzialmente diverso da quello proposto qui.

[5] Si veda, su questi temi, il cap. 2 di Meneghelli (2007).

[6] Queste traduzioni verrano prese in considerazione in seguito. Sulla questione dei concatenamenti tra diverse enunciazioni si veda Fontanille (2006).

[7] Con *tab* ci si riferisce alle schede di navigazione utilizzate da diversi browser (Opera, Firefox, ora anche Explorer) per consentire la navigazione in contemporanea su più pagine web.

[8] A riprova di ciò nella visualizzazione split, la metà codice è posta prima sopra quella design (fig. 3).

Bibliografia

Ducheneaut, N., 2005, "Socialization in an Open Source Software Community: A Socio-Technical Analysis", in *Computer Supported Cooperative Work*, 14.

Fontanille, J., 2004, *Figure del corpo*, Roma, Meltemi.

Fontanille, J., 2006, "Affichages: de la sémiotique des objets à la sémiotique des situations", in *E/C, Rivista on-line dell'Associazione Italiana di Studi Semiotici* (www.ec-aiss.it).

Laurel, B. e Mountford, S. J., 1990, "Introduction" in B. Laurel, a cura, *The art of human-computer interface design*, Reading MA, Addison-Wesley, 1990, pp. XI-XXI.

Mattozzi, A., 2009, "A Model for the Semiotic Analysis of Objects" in S. Vihma e T-M. Karjalainen, a cura, *Design Semiotic in Use*, Helsinki, Helsinki University of Art and Design Press, in corso di pubblicazione.

Meneghelli, A., 2007, *Dentro lo schermo. Immersione e interattività nei god games*, Milano, Unicopli.

Nichols, D., Twidale, M. 2003, "The Usability of Open Source Software", in *First Monday*, 8 (1).

Raymond, E. S., 1998, "The cathedral and the Baazar", in *First Monday*, 3,3.