UNIVERSITÀ DEGLI STUDI DI PISA
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

CORSO DI LAUREA SPECIALISTICA IN TECNOLOGIE INFORMATICHE

Sistema di adattamento automatico di applicazioni interattive desktop per dispositivi mobili

Relatore:
Dott. Fabio Paternò

Candidato:
Agazio Gregorace

ANNO ACCADEMICO 2006–2007

Alla mia famiglia

Sommario

Il lavoro di tesi presenta un sistema, basato su un proxy server, per l'adattamento automatico di pagine web all'accesso tramite dispositivi mobili. Il sistema esegue il processo di trasformazione prendendo in considerazione la descrizione logica delle pagine ed il costo in termini di spazio occupato dagli elementi che compongono l’interfaccia utente (testo, immagini, bottoni, ecc).

Indice

Capitolo 1

Introduzione

1.1 Introduzione

L'utilizzo di Internet è aumentato enormemente negli ultimi anni determinando un incremento quasi esponenziale del contenuto web presentato agli utenti. In particolare gli sviluppi recenti hanno fatto registrare un aumento dell'uso di applicazioni di rete e di dispositivi mobili.

Da un paio d'anni, questi ultimi (terminali mobili, smartphones, PDA, ecc), sono diventati rapidamente una parte integrante della nostra vita. Attraverso di essi, possiamo parlare, mandare messaggi o fare video chiamate, e con una connessione internet possiamo accedere a qualsiasi tipo di contenuto web.

Tuttavia, l'accesso a Internet tramite i dispositivi è limitato dal momento che la maggior parte dei contenuti web sono creati appositamente per computer desktop.

Paragonati ai computer desktop, i dispositivi mobili hanno qualche evidente inconveniente. La maggior parte di essi hanno risorse limitate e sono connessi a reti con banda limitata. Inoltre, le loro caratteristiche variano normalmente in termini di capacità e di tipo di memoria, di risoluzione, di velocità del processore e di banda per il download.

Le prestazioni dei dispositivi mobili sono di solito molto basse, i display sono piccoli. Per esempio, la dimensione dello schermo per un Personal Computer (PC) varia tra 800x600 e 1800x1440 pixel, mentre per un PDA tra 160x160 e 320x240 pixel e per un terminale mobile (smartphone o dispositivi con display relativamente grandi) tra 128x128 e 176x220 pixel. Questi ultimi, inoltre, utilizzano una varietà di connessioni di rete che vanno dal cavo di rete al wireless, con banda, caratteristiche di connessione e costi variabili. Di solito, infatti, le connessioni per dispositivi mobili sono molto più lente e più costose rispetto alle altre. Tuttavia, al giorno d'oggi, pur essendo in crescente aumento il numero di persone che preferisce usare un terminale mobile per accedere al ricco contenuto web attraverso Internet,

la maggior parte degli sviluppatori web sono ancora principalmente indirizzati ai personal computer con schermi grandi, capaci, cioè, di visualizzare quasi per intero il contenuto delle pagine web.

Uno dei maggiori limiti di molti dispositivi mobili è rappresentato dal fatto che il loro display non è capace di supportare tutta la ricchezza del contenuto web, per cui alcune volte accade che il contenuto e il layout delle pagine web, spesso molto complessi, non si adattino ai piccoli display richiedendo, quindi, un considerevole scrolling orizzontale e verticale da parte dell'utente, mentre altre volte le pagine web verrebbero deformate, nel momento in cui il browser cerca di adattarle al display del dispositivo mobile. Tutto questo rende la navigazione web su questi dispositivi un'attività noiosa e stancante.

Il problema della navigazione web attraverso dispositivi mobili ha attirato l'attenzione sia della comunità della ricerca sia dell'industria. Entrambe ritengono necessario sviluppare metodi e tecniche per adattare il contenuto originario alle capacità dei dispositivi mobili.

In questa tesi verrà presentato un sistema per l'adattamento automatico di pagine web.

La soluzione proposta è di usare un proxy server che trasforma *on-the-fly* una pagina web desktop in modo tale che possa essere visualizzata in maniera adeguata sul display di un dispositivo mobile.

Il proxy, situato tra il dispositivo mobile (client) e il server web che contiene la pagina web richiesta, intercetta tutte le richieste del dispositivo mobile e prima di restituire il contenuto richiesto esegue delle trasformazioni sulla pagina originaria.

Il ruolo principale del sistema consiste nel processo di trasformazione. Il suo input è una pagina web (X)HTML originariamente realizzata per piattaforma desktop e il suo output è un insieme di pagine adattate al display del dispositivo mobile che corrispondono a quella originaria.

Il processo di trasformazione prende in considerazione gli aspetti semantici e il costo in termini di spazio occupato sulla pagina dagli elementi che compongono l'interfaccia utente (testo, immagini, bottoni, ecc).

Basandosi sull'idea del proxy server, verrà proposta una soluzione che fa uso della descrizione logica della pagina web, per riadattarla alle risorse del dispositivo mobile destinatario.

La descrizione logica, specificata utilizzando un linguaggio basato su XML, è usata principalmente per estrarre le informazioni semantiche dalla pagina web. Sfruttando queste informazioni è possibile trasformare il contenuto e il layout della pagina web così che ad essa si possa accedere attraverso un qualsiasi dispositivo mobile.
Fondamentalmente il processo di trasformazione consiste di tre fasi principali: *Reverse*, *Redesign* e *Pages Generator*.
La fase di *Reverse* esegue un processo di *reverse engineering* della pagina web desktop creando la descrizione logica della stessa in un linguaggio basato su XML. Il risultato viene trasformato dal modulo *Redesign* applicando un insieme di regole di trasformazione che permettono così di ottenere la descrizione logica dell'interfaccia utente per il dispositivo mobile. Infine, la fase di *Pages Generator* genera le corrispondenti pagine web adattate al display del dispositivo mobile destinatario.
Il sistema è stato realizzato in modo tale da poter essere applicato in futuro anche ad altri dispositivi quali PDA, dispositivi vocali, ecc.
Con il presente lavoro ci si propone in sostanza di:

- riutilizzare contenuti web senza doverli riprogettare da zero per adattarli alle risorse dei dispositivi mobili, al fine di ridurre i costi e gli sforzi necessari per sviluppare contenuti web per dispositivi mobili.

- generare delle interfacce utente adatte a poter accedere facilmente contenuti web con terminali mobili, attraverso una tecnica per navigare le pagine web con controlli tipo bottoni/link. A tal fine, la struttura originaria della pagina è modificata, sia eliminando lo scrolling orizzontale sia dividendo la pagina in più pagine (*splitting*) per ridurre lo scrolling verticale e poterla così adattare al display del dispositivo mobile.

1.2 Organizzazione della tesi

Nel secondo capitolo viene fornita una panoramica dello stato dell'arte e una rassegna delle soluzioni esistenti.

Nel terzo capitolo viene presentato l'algoritmo di trasformazione delle pagine web (*Semantic Redesign*) pre-esistente e le regole principali utilizzate per trasformare le pagine web per adattarle ai dispositivi mobili.

Nel quarto capitolo vengono presentati il nuovo algoritmo e i relativi aspetti progettuali.

Nel quinto capitolo viene fornita una descrizione dell'architettura del prototipo di sistema che è stato realizzato.

Infine nel sesto capitolo vengono esposte le conclusioni e gli sviluppi futuri.

Capitolo 2

Stato dell’arte e soluzioni esistenti

2.1 Approcci per presentare contenuti web nei dispositivi mobili

Bickmore [12] dà una classificazione abbastanza soddisfacente delle tecniche per presentare contenuti web nei dispositivi con piccoli display. Egli distingue cinque categorie: *device-specific authoring*, *multiple-device authoring*, *client-side navigation*, *web page filtering* e *automatic re-authoring*.

Con i primi due approcci si ottengono risultati di alta qualità dal momento che è possibile creare contenuti web per ogni specifico dispositivo.

Avere due versioni dello stesso contenuto web (una per il desktop e l’altra per il dispositivo mobile) non solo è costoso ma non costituisce neanche un approccio usato dalla maggior parte dei siti web, per cui Internet risulta in gran parte inaccessibile agli utenti dei dispositivi mobili. Le altre tecniche al contrario non richiedendo la collaborazione degli autori delle pagine web sono maggiormente applicabili.

Di seguito verranno descritte queste categorie.

2.1.1 Device-specific authoring

Device-specific authoring, detto anche *manual authoring*, significa creare un’applicazione web avendo in mente il dispositivo a cui è rivolta.

Le pagine che vengono realizzate in questo modo saranno sì perfettamente visualizzabili sul display del dispositivo destinatario, ma questo può visualizzarne soltanto un numero limitato in maniera adeguata.

L'approccio, quindi, oltre ad essere inefficiente è anche molto costoso, dal momento che il contenuto web deve essere presentato separatamente per ogni dispositivo.
A causa del grosso lavoro manuale che richiede, questa tecnica può essere applicata soltanto ad un insieme ridotto di siti web ed inoltre è limitata in scalabilità.
Un esempio di *device-specific authoring* è Amazon.com[1] che fornisce differenti versioni del sito web, ad esempio, per terminali mobili WAP e PDA (figura 2.1).

Figura 2.1: Home page di *Amazon* per dispositivi mobili

2.1.2 Multiple-device authoring

Multiple-device authoring è relativamente simile al *device-specific authoring*, solo che in questo caso l'applicazione web stessa è realizzata per supportare un insieme ben definito di dispositivi e il contenuto presentato all'utente è formattato in base al tipo di dispositivo che fa la richiesta.

[1] http://www.amazon.com/anywhere

Anche questo approccio, come il precedente, richiede un considerevole lavoro manuale per la realizzazione delle pagine web.
Un esempio di *multiple-device authoring* è offerto dai *cascading style sheets* (CSS), nei quali un singolo *style sheet* definisce un insieme di attributi di visualizzazione per le diverse parti delle pagine web.

2.1.3 Client-side navigation

Con il *client-side navigation* la pagina web viene trasferita nel dispositivo mobile senza nessuna modifica ed è il browser del dispositivo che si occupa poi della sua presentazione.
Data una pagina web, l'utente può navigare interattivamente o utilizzando la barra di scrolling ai lati dell'area del display (approccio adottato dalla maggior parte dei browser di molti PDA) oppure "zoomando" la parte della pagina che interessa.
Quest'ultimo approccio è usato in [13], in cui l'utente può "ridurre ed espandere" la parte della pagina che gli interessa. In particolare questo permette all'utente di "ridurre" l'area ritenuta irrilevante come per esempio menu o pubblicità e di espandere il resto per identificare l'informazione che interessa.

2.1.4 Web page filtering

Web page filtering permette all'utente di visualizzare soltanto le informazioni desiderate attraverso un processo di filtraggio.
Questo può avvenire sia sul server che sul dispositivo mobile (client) ed utilizza solitamente qualche database in cui l'utente può memorizzare un profilo contenente, per esempio, delle parole chiavi.

2.1.5 Automatic re-authoring

Automatic re-authoring è il metodo più diffuso per l'adattamento delle pagine web per dispositivi mobili che prende in considerazione le caratteristiche del dispositivo e automaticamente, *on-the-fly*, riformatta la pagina web richiesta per poterla adattare alle risorse del dispositivo destinatario.

In generale, il metodo richiede un'interazione minima da parte dell'utente e comporta la realizzazione di un'architettura software per riformattare una pagina web attraverso una serie di trasformazioni in tempo reale, così che essa possa essere visualizzata in maniera appropriata sul display del dispositivo mobile destinatario.

L'approccio può essere a sua volta classificato in due ulteriori classi [14, 15]: *transducing* e *transforming*.

Transducing traduce l'HTML in altri formati (ad esempio da HTML a WML, cHTML, HDML) e, comprime e converte le immagini per farle adattare alle risorse del dispositivo (un esempio è costituito da AvantGo[2]).

Transforming modifica sia il contenuto che la struttura delle pagine web originariamente progettate per essere navigate con dispositivi desktop per renderle adatte ad essere visualizzate su piccoli display.

Con questa tecnica si modifica il layout delle pagine web, "splittandole" in più sotto-pagine, e si aggiungono nuovi link per navigare queste ultime.

In base al posto in cui l'adattamento viene eseguito si possono distinguere tre classi [16, 17]: *server-side*, *client-side* o *proxy-side adaptation* (figura 2.2).

[2] http://www.avantgo.com

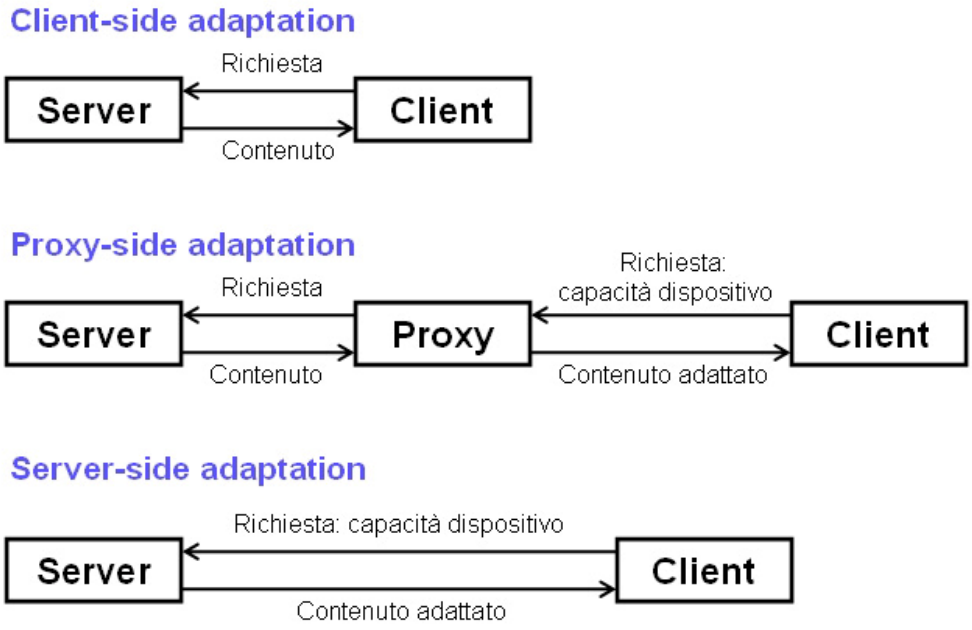


Figura 2.2: Differenti posti di adattamento delle pagine web

Server-side adaptation

Server-side adaptation fa si che il server restituisca ai client delle pagine web già adattate rispetto a quelle originarie. Ciò avviene o attraverso un adattamento dinamico *on-demand* o avendo un database di pagine web già adattate.

Server-side adaptation è svolto o dal fornitore di servizi o dal portale dell'operatore mobile che fornisce il contenuto esclusivamente per dispositivi mobili. Inoltre, offre agli autori delle pagine web il massimo controllo su come presentarle.

Gli autori possono usare diversi metodi di presentazione [16, 18]:

- *Single authoring*: viene creata una sola versione della pagina web che poi è presentata dal browser stesso in un formato adatto al display del dispositivo.
- *Multiple authoring*: vengono create differenti versioni delle pagine web per ogni browser dei dispositivi.

Client-side adaptation

Nel *client-side adaptation* o *browser adaptation* l'adattamento, e di conseguenza la presentazione finale della pagina web, è eseguita dal dispositivo client.
Il più semplice e probabilmente il più usato esempio di *client-side adaptation* è rappresentato dall'uso dei CSS.
Questi permettono di separare lo stile dal contenuto, così che gli autori delle pagine web possano fornire stili diversi a differenti dispositivi. Ad esempio il browser Opera[3] usa una tecnologia chiamata *Small-Screen Rendering*[4] per riformattare il contenuto della pagina web originaria (riduce i font e le immagini) e crea un layout lungo e stretto (*narrow layout*) per eliminare lo scrolling orizzontale, mentre quello verticale aumenta in modo considerevole.
Il *client-side adaptation* presenta però alcuni inconvenienti: molti dispositivi mobili hanno, infatti, bassa potenza di calcolo, poca memoria e banda limitata.
Il processo di adattamento richiede di norma operazioni computazionali complesse e molto tempo, ma fortunatamente la potenza di calcolo dei dispositivi mobili migliora continuamente.

Proxy-side adaptation

Nel *proxy-side adaptation* un proxy server analizza e trasforma il contenuto *on-the-fly* prima di mandare il risultato al client.
Il proxy server può anche mantenere in cache il contenuto adattato per usi futuri, in modo che la trasformazione avvenga una sola volta. Con questo approccio si può spostare il carico computazionale dal server al proxy [19] che però deve conoscere il tipo di dispositivo che fa la richiesta per poter restituire il contenuto esatto.
I metodi basati su proxy utilizzano in genere regole euristiche per eseguire la trasformazione [18].

[3] http://www.opera.com
[4] Opera Small Screen Rendering. http://www.opera.com/products/smartphone/smallscreen/

Il motore di ricerca Google possiede una versione per dispositivi mobili [20]. Il proxy di Google[5] trasforma le pagine richieste dall'utente in un layout stretto e compresso: viene eliminato lo scrolling orizzontale e una singola pagina può essere "splittata" per ridurre lo scrolling verticale.

Un altro esempio è il browser Opera Mini[6] per terminali mobili che è in grado di riformattare le pagine web per adattarle ai display dei dispositivi (figura 2.3).

Il browser Opera Mini, prima di presentare la pagina sul dispositivo, contatta il proxy server di Opera, che prendendo la pagina richiesta, la filtra da quanto c'è di superfluo (animazioni, immagini di troppo, ecc.), la comprime e la invia al terminale mobile che avrà così un minor carico di lavoro e potrà quindi visualizzare i dati più rapidamente [16].

[5] http://www.google.com/xhtml

[6] Opera Mini Browser. http://www.opera.com/products/mobile/operamini/

Figura 2.3: *Narrow layout* nel browser Opera Mini per dispositivi mobili

Oltre agli esempi appena illustrati vi sono anche Mowser[7] e Skweezer[8] che si basano sempre su un proxy server per riformattare *on-the-fly* siti web così che si possano adattare ai display dei dispositivi mobili.

[7] http://mowser.com
[8] http://www.skweezer.net

2.2 Tecniche e sistemi esistenti per l'adattamento del contenuto web

Qui di seguito verranno descritte alcune delle tecniche e dei sistemi esistenti per l'adattamento del contenuto web per dispositivi mobili che sono tuttora oggetto di ricerca.

I primi lavori puntavano su WML (Wireless Markup Language) e WAP (Wireless Application Protocol) per progettare e visualizzare pagine web sui browser dei dispositivi mobili, ma questi approcci imponevano sforzi aggiuntivi ai web designer dal momento che dovevano creare contenuti WML appositi [21].

Ricerche successive si concentrarono sulla possibilità di adattare dinamicamente il contenuto web ai dispositivi mobili con piccoli display.

Dalle ricerche fatte da [22], è emerso che il primo lavoro che ha affrontato direttamente il problema del bisogno di tool automatici per l'adattamento del contenuto e del layout è il progetto Digestor [23], messo in atto nel 1997 da Bickmore e il suo gruppo.

Bickmore classifica le tecniche di adattamento automatico in due categorie:

- Categoria sintattica: opera sulla struttura della pagina web.
- Categoria semantica: dipende dalle stesse conoscenze semantiche.

Egli, inoltre, presenta diverse proposte su come adattare il contenuto web.

Digestor, implementato come un proxy server, nello specifico utilizza un insieme di tecniche di re-authoring che dipendono da priorità e condizioni al fine di risparmiare lo spazio nel display. Inoltre, fa si che una pagina web possa essere "splittata" in sotto-pagine (ognuna delle quali si adatta bene al display del dispositivo mobile) e che nuovi link di navigazione vengano aggiunti per navigare tra le pagine "splittate".

Altre ricerche concentrano l'attenzione sulle tecniche di *summarization* [24] che cercano di "riassumere" automaticamente il contenuto delle pagine web estraendo l'informazione importante per presentarla ai dispositivi con piccoli display e memoria limitata.

Di solito queste tecniche si basano su delle regole che "guidano" l'estrazione e il *summarize*.

Tecniche di *page splitting*, usate nell'adattamento del contenuto web per dispositivi mobili, sono descritte in [15, 25].

Una tecnica di *page analysis* viene proposta in [25] per analizzare la struttura di una pagina web e "splittarla" in piccole unità logicamente correlate che si adattano al display di un dispositivo mobile. Nel caso in cui la pagina non è adatta per essere "splittata", un metodo di auto-posizionamento o scrolling a blocco viene usato per facilitare la navigazione.

In [15], una pagina web, che risulta essere grande per il piccolo display, è trasformata in un insieme di pagine, ognuna delle quali si adatta al display.

Altri prototipi di sistemi che utilizzano *automatic re-authoring* si inseriscono in due categorie principali: *page reformatting* e *page scaling*.

Esempi di tecniche basate sul *page reformatting* sono Power Browser [26] e il WEST [27]. Il primo divide una pagina web in unità testuali semantiche e crea un sommario basato sulle unità. Il secondo propone uno schema chiamato "*focus + context*" che fornisce una descrizione della pagina web. Questi due approcci hanno un impatto notevole sul layout della pagina adattata.

Altre tecniche basate sempre sul *page reformatting* prevedono il ridimensionamento della pagina e la restituzione del risultato al dispositivo client come un'anteprima della pagina in dimensioni ridotte *(thumbnail*).

Un esempio di questo modo di operare è costituito dal sistema SmartView [28] che crea un'immagine della pagina di dimensioni ridotte (*trumbnail*) prendendo in considerazione principalmente il suo contenuto semantico. L'utente ha la possibilità di zoomare la parte dell'anteprima della pagina che gli interessa e di leggerne il suo contenuto. Il problema principale di questo approccio è rappresentata dalla dimensione dell'anteprima della pagina: infatti, la dimensione ridotta del display del dispositivo mobile non permette in molti casi che il testo sia leggibile.

Un altro approccio cerca, invece, di risolvere questo problema combinando il metodo precedente con le tecniche di *text summarization*.

L'idea principale su cui si basa questo modo di operare [29] è che l'anteprima della pagina contenga soltanto alcuni frammenti di testo e che l'utente poi zoomando un'area specifica abbia la possibilità di leggere l'intero testo (figura 2.5).

Figura 2.5: Esempio di *Summary thumbnail*

L'ultimo metodo è principalmente destinato agli utenti di dispositivi PDA e di terminali mobili con display relativamente grandi, dal momento che quelli con piccoli display non sono capaci di presentare una veduta soddisfacente dell'anteprima della pagina.

Infatti, la risoluzione tipica del display di un terminale mobile varia tra 128x128 e 176x220 pixel. Dal momento che una generica pagina web, ricca di contenuto, richiede uno scrolling anche con un monitor di 17 pollici per essere visualizzata per intero, risulta ovvio che l'anteprima della pagina visualizzata sul terminale mobile sarà così piccola da risultare poco leggibile all'utente, specialmente se è la prima volta che visita la pagina.

2.2 Approccio basato su automatic re-authoring, XML e regole semantiche

L'approccio presentato in questa tesi può essere classificato nella categoria *automatic re-authoring* con *proxy-side adaptation*. Esso si basa, inoltre, su una *descrizione logica XML* e *regole semantiche*.

Nostro approccio:

Automatic re-authoring + descrizione logica XML + regole semantiche

Il processo di trasformazione è mostrato in figura 2.6.

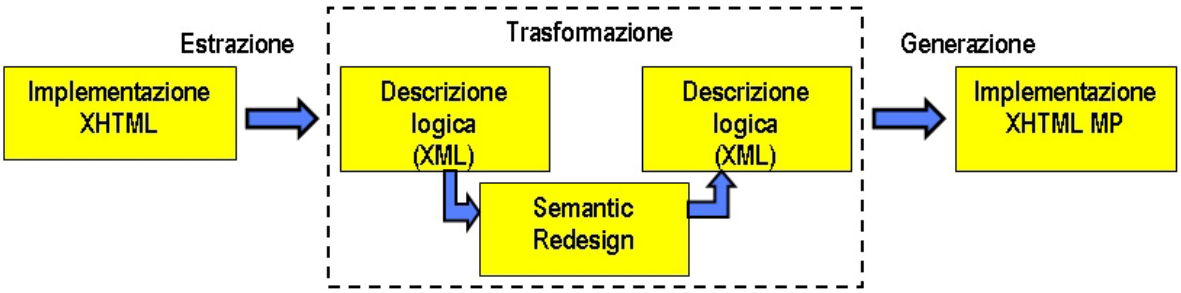


Figura 2.6: Processo di trasformazione attraverso l'approccio adottato

Automatic re-authoring

Tra tutti gli approcci esistenti per la presentazione di pagine web su dispositivi mobili, *automatic re-authoring* è quello che fornisce più flessibilità e largo accesso ai contenuti web per un'ampia gamma di dispositivi.

Automatic re-authoring può essere messo in atto modificando alcuni aspetti del contenuto o della presentazione della pagina, oppure rimuovendo qualche informazione non importante e lasciando il resto intatto.

Descrizione logica XML

Come si è detto precedentemente l'approccio utilizza una descrizione logica basata su XML per generare le pagine web per i dispositivi mobili.
Lavorare con contenuti basati su XML ha raggiunto una grande popolarità negli ultimi anni [11].
La ragione principale è che XML separa il contenuto dalla presentazione, per cui possiamo disporre di molte descrizioni del layout per lo stesso documento ed il layout può essere generato dinamicamente. Oltre a ciò è più facile convertire documenti XML in altri formati come XHTML, per cui risulta ancor più idoneo l'utilizzo di soluzioni di *automatic re-authoring*.
Nel nostro approccio, la trasformazione dell'interfaccia utente dell'applicazione desktop è basata sull'uso di una descrizione logica, mediante la quale, applicando delle trasformazioni, è possibile ottenere l'interfaccia utente finale per il dispositivo mobile, mantenendo le caratteristiche semantiche di quella originaria.

Regole semantiche

Molte delle soluzioni alle quali abbiamo accennato precedentemente trasformano pagine web che sono state originariamente progettate per una specifica risoluzione dello schermo, per cui, pur essendo molto efficienti, non potranno mai essere perfette, dal momento che le pagine web sono state progettate senza presupporre una loro futura trasformazione. In qualche modo, l'algoritmo di trasformazione dovrebbe conoscere il ruolo svolto da ciascun elemento presente nella pagina web per adattarlo in maniera efficiente.
Sfortunatamente la maggior parte dei linguaggi di markup come HTML non permettono di fornire questa informazione extra, per cui molti di questi metodi si basano su informazioni euristiche.
Al contrario, le informazioni contenute nella descrizione logica della pagina web, ci forniscono informazioni semantiche che possono essere utili al fine di identificare modi migliori per "splittare" la pagina web desktop.

Capitolo 3

Progettazione di interfacce basate su modelli

3.1 Introduzione

La crescente disponibilità di diversi tipi di dispositivi mobili pone una serie di sfide ai progettisti e agli sviluppatori di applicazioni interattive.

La maggior parte delle applicazioni internet sta diventando accessibile attraverso l'uso di molteplici dispositivi. Tuttavia, per garantire agli utenti una esperienza soddisfacente dal punto di vista dell'usabilità, è importante che l'interfaccia utente dell'applicazione sia in grado di adattarsi alle risorse supportate da ciascun tipo di dispositivo.

Avere uno sviluppo separato di interfacce utente per ogni ipotetico dispositivo è piuttosto costoso.

Questo ha fatto crescere l'interesse nel campo della ricerca di approcci basati su modelli per la realizzazione di applicazioni interattive che forniscano descrizioni logiche dell'interfaccia utente e che siano supportati da tool che generano l'implementazione corrispondente prendendo in considerazione le caratteristiche del dispositivo destinatario.

L'approccio basato su modelli sembra adatto a gestire nel modo migliore la complessità che si trova nelle applicazioni interattive multi-dispositivo, perché queste ultime permettono lo sviluppo di modelli logici e di trasformare successivamente e di adattare i risultati a differenti piattaforme.

Così, cresce il bisogno di tool capaci di supportare ciò.

Un esempio è costituito da TERESA (**T**ransformation **E**nvironment for inte**R**activ**E** **S**ystems represent**A**tion) [10] che fornisce un supporto per diversi metodi basati su modelli ed è un

ambiente di *authoring* per la progettazione e la generazione automatica di interfacce utente multi-dispositivo.
In questo capitolo viene fornita un'introduzione dei linguaggi basati su XML (TERESA XML) usati da TERESA e descritto l'algoritmo di *automatic semantic platform-dependent redesign* che il tool supporta [1].
Quest'ultimo è utile, per esempio, nella realizzazione di applicazioni interattive per dispositivi mobili, in particolare quando esiste già una versione desktop dell'applicazione e successivamente il progettista vorrebbe ottenerne una versione per dispositivi mobili, capace di adattarsi alle loro risorse, riutilizzando gli elementi già usati nella versione desktop.

3.2 Concetti di base

TERESA XML sviluppato dal gruppo HCI[9] del ISTI-CNR ipotizza che ci possono essere diversi livelli di astrazione dell'interfaccia utente di un'applicazione interattiva (figura 3.1):

- *Modello dei Task (CTT)*: a questo livello vengono considerate le attività logiche compiute dagli utenti (non trattato in questa tesi).
- *Interfaccia Utente Astratta (AUI)*: a questo livello vengono presi in considerazione gli oggetti di interazione astratti necessari al completamento delle varie attività mentre vengono trascurati gli aspetti implementativi.
- *Interfaccia Utente Concreta (CUI)*: in relazione al tipo di dispositivo preso in esame viene considerato un opportuno linguaggio basato su XML per l'implementazione dell'interfaccia utente. Per ciascun oggetto di interazione astratto viene determinato, tra quelli messi a disposizione dal linguaggio scelto, l'interattore concreto più opportuno alla sua implementazione.
- *Interfaccia Utente Finale (FUI)*: l'interfaccia utente vera e propria, codificata nel linguaggio di implementazione desiderato.

[9] http://giove.cnuce.cnr.it

Per ognuno dei primi tre livelli [2] definisce uno specifico linguaggio basato su XML.

Dal momento che il livello astratto più basso (l'interfaccia concreta) è dipendente dalla piattaforma vi sono differenti varianti in relazione alla piattaforma considerata (es. desktop e mobile).

Nei paragrafi successivi verrà fornita una panoramica della AUI e in particolare delle CUI che sono state utilizzate per generare l'interfaccia utente per i dispositivi mobili.

Verrà dato solo un accenno alla descrizione astratta, in quanto le informazioni che la riguardano sono integrate nella descrizione concreta che è un raffinamento che aggiunge informazioni riguardanti attributi concreti alle strutture fornite dalla descrizione astratta.

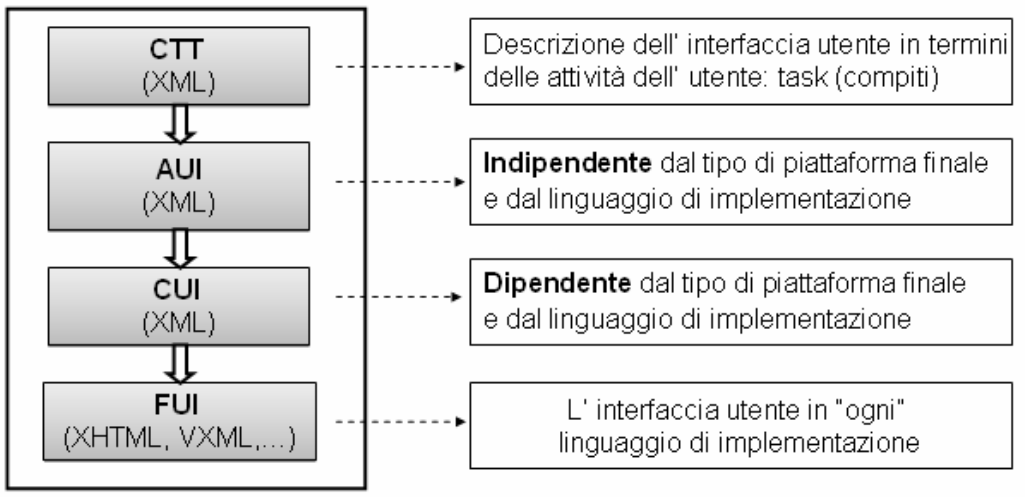


Figura 3.1: Livelli di astrazione di una interfaccia utente e descrizioni logiche

3.2.1 Linguaggio per la rappresentazione dell'*Interfaccia Utente Astratta*

La AUI (Interfaccia Utente Astratta) permette di descrivere un'interfaccia astratta in termini degli elementi di interazione che la compongono e del suo comportamento in relazione alle interazioni degli utenti con tali oggetti.
Di seguito verranno descritte le caratteristiche principali e la struttura del linguaggio mediante l'analisi della sua DTD.

3.2.1.1 Descrizione generale dell'AUI

L'interfaccia utente astratta è composta da un certo numero di presentazioni[10] che ne rappresentano la struttura statica e da un certo numero di connessioni che indicano come è possibile muoversi da una presentazione ad un'altra.
Queste ultime in particolare definiscono la caratteristica dinamica dell'interfaccia utente.
Ogni presentazione è composta da interattori e operatori di composizione.
In [2] sono stati definiti alcuni operatori di composizione che cercano di catturare gli obiettivi di comunicazione che i web designer si prefiggono nel momento in cui progettano le interfacce web. Lo scopo degli operatori di composizione è di indicare come mettere insieme gli interattori. Ogni operatore di composizione è associato ad un obiettivo di comunicazione.
La figura 3.2 mostra una generica pagina presa da un sito web e alcuni operatori di composizione.

[10] Per ogni pagina web XHTML corrisponde una "presentazione" nel linguaggio TERESA XML

Figura 3.2: Pagina web con l'indicazione di alcuni obiettivi di comunicazione associati

Dalla figura emerge che il web designer ha impiegato varie tecniche per raggruppare gli elementi correlati dell'interfaccia web: alcuni sono ordinati secondo l'interesse dell'utente, altri sono raggruppati utilizzando delle tecniche di implementazione (come ad esempio stesso sfondo, stessa struttura, bullet, ecc) altri ancora sono correlati con il resto del sito web (come l'elemento *search*), infine, altri elementi sono evidenziati usando delle immagini larghe o font grandi perché essi sono considerati importanti.

In generale, gli operatori di composizione possono coinvolgere sia interattori sia composizione di interattori e di operatori di composizione stessi.

Gli operatori di composizione che possono essere applicati agli interattori sono:

- **Grouping (G):** l'applicazione di questo operatore consente di raggruppare tra loro due o più elementi dell'interfaccia. Generalmente, viene applicato quando i task coinvolti condividono alcune caratteristiche strutturali.

- **Ordering (O):** l'applicazione di questo operatore consente di stabilire un certo tipo di ordine tra gli elementi ai quali viene applicato.
- **Relation (R):** si può applicare questo operatore quando un certo numero di elementi dell'interfaccia astratta sono in relazione con un altro elemento.
- **Hierarchy (H):** questo operatore stabilisce l'esistenza di una gerarchia tra gli interattori ai quali viene applicato.

Ci sono differenti tipi di elementi di interazione che dipendono dal tipo di task supportato. In particolare viene fatta una distinzione tra interattori che supportano l'interazione con l'utente (*interaction element*) e quelli che presentano risultati derivanti dall'elaborazione di applicazioni (*only output element*).
I primi implicano un'interazione tra utente e applicazione. Ne esistono diversi tipi: *select* (per selezionare tra un insieme di elementi), *edit* (per editare un campo), *control* (per lanciare un evento all'interno dell'interfaccia, che può essere utile sia per attivare una funzionalità che per spostarsi in una nuova presentazione).
Esistono anche diversi elementi *only output* (*text*, *object*, *description*, *feedback*) dipendenti dal tipo di output che l'applicazione fornisce all'utente: testuale, un oggetto, una descrizione, un feedback circa un particolare stato dell'interfaccia web.
La AUI è completamente indipendente dalla piattaforma e dal linguaggio di implementazione e fornisce indicazioni su quali tipi di oggetti sono necessari per supportare l'esecuzione di un certo task e su come questi devono essere composti nelle presentazioni.

3.2.1.2 DTD per il linguaggio di specifica della AUI

Nel diagramma in figura 3.3 viene mostrata la struttura ad albero degli elementi del linguaggio come definiti nella DTD.

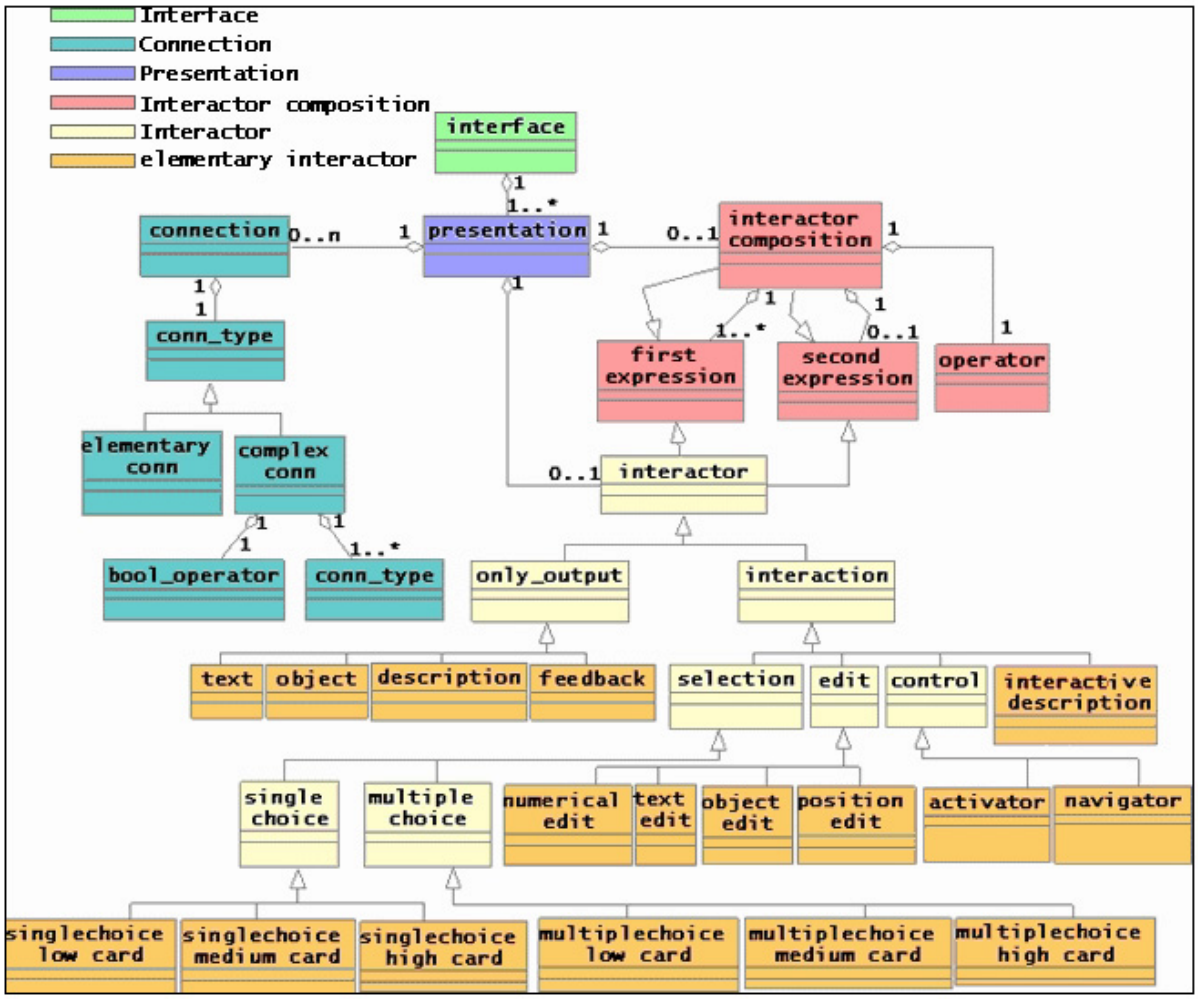


Figura 3.3: Struttura della DTD dell'Abstract User Interface (AUI)

3.2.2 Linguaggio per la specifica dell'*Interfaccia Utente Concreta*

L'Interfaccia Utente Concreta (CUI) arricchisce la descrizione dell'AUI aggiungendo ulteriori dettagli implementativi agli oggetti del livello astratto (per esempio, può specificare il font, il colore o lo stile di un elemento testuale).

3.2.2.1 DTD del linguaggio per le piattaforme desktop e mobile

In questa sezione viene descritto il linguaggio XML per le due piattaforme utilizzate: desktop e terminale mobile.
In entrambi i casi la struttura è simile a quella del linguaggio dell'AUI.
I dettagli implementativi sono per lo più forniti dai livelli più bassi di una rappresentazione ad albero i cui nodi foglia definiscono le informazioni concrete.
Le differenze principali delle CUI dei dispositivi possono essere reperite nei livelli più bassi dell'albero.
Un'interfaccia utente concreta desktop è definita da un certo numero di presentazioni e di impostazioni di default. Queste ultime in particolare sono utilizzate nella fase di generazione delle pagine.

```
<!ELEMENT concrete_desktop_interface (default_settings,
                                      presentation+)>


<!ELEMENT default_settings (background,
                            font_settings,
                            operators_settings,
                            interactors_settings)>
```

I terminali mobili richiedono altre informazioni riguardanti le loro capacità.

```
<!ELEMENT concrete_mobile_interface (device_type,
                                     default_settings,
                                     presentation+)>

<!ELEMENT device_type (big | medium | small)>

<!ELEMENT big EMPTY>
<!ATTLIST big
      graphic_support (%option;) #REQUIRED>

<!ELEMENT medium EMPTY>
<!ATTLIST medium
      graphic_support (%option;) #REQUIRED>

<!ELEMENT small EMPTY>
<!ATTLIST small
      graphic support CDATA #FIXED "no">
```

Ogni presentazione contiene informazioni sull'organizzazione degli interattori, le connessioni verso altre presentazioni e fornisce una descrizione di proprietà specifiche quali titolo, header, sfondo, ecc.

```
<!ELEMENT presentation (presentation_properties,
                        connection*,
                       (interactor | interactor_composition))>

<!ELEMENT presentation_properties (title,
                                   background,
                                   font_settings,
                                   top)>
```

I terminali mobili prevedono un sottoinsieme delle proprietà previste per i sistemi desktop.

```
 <!ELEMENT presentation_properties (title, top?)>
```

Per ogni tipo di piattaforma sono definite le connessioni e gli operatori di composizione come nella AUI.

```
<!ELEMENT connection (conn_type)>
<!ATTLIST connection
      presentation_name IDREF #REQUIRED>

<!ELEMENT conn_type (elementary_conn | complex_conn)>
[…]

<!ELEMENT interactor (interaction | only_output)>

<!ELEMENT interactor_composition (operator, first_expression+,
            second_expression?)>

<!ELEMENT operator (grouping | ordering | hierarchy | relation)>
<!ATTLIST operator
      id ID #REQUIRED>
```

Andando sempre più nel dettaglio nella specifica degli interattori, vediamo, per esempio, come sia possibile descrivere un interattore *single selection* con la notazione CUI-desktop.

```
<!ELEMENT interaction ( selection | editing | control |
                        interactive_description )>

<!ELEMENT selection (single | multiple)>

<!ELEMENT single (radio_button | list_box | drop_down_list)>
<!ATTLIST single
      cardinality (%cardinality_value;) #REQUIRED>

<!ELEMENT radio_button (choice_element+)>
<!ATTLIST radio_button
      label CDATA #REQUIRED
      alignment (%element_selection_alignment;) #REQUIRED>

<!ELEMENT choice_element EMPTY>
<!ATTLIST choice_element
      label CDATA #REQUIRED
      value CDATA #REQUIRED>

[…]
```

Le maggiori differenze tra le CUI si riscontrano negli elementi concreti associati a un dato tipo di interattore.
Se consideriamo l'esempio del *single selection*, visto prima per il terminale mobile, si vede che in un terminale mobile i *list box* non sono idonei e non sono quindi permessi.

```
<!ELEMENT single (radio_button | drop_down_list)>
```

Ogni CUI formalizza il potere espressivo di un dato tipo di dispositivo in termini di interattori concreti e di operatori di composizione disponibili in quella piattaforma.
Mentre nel caso della piattaforma desktop l'operatore di *grouping* può essere implementato combinando molte tecniche, nei terminali mobili è possibile scegliere solo una tecnica di implementazione.

CUI-desktop

```
<!ELEMENT grouping (fieldset?, bullet?, background_color?, position)>
```

CUI-mobile

```
<!ELEMENT grouping ((fieldset | bullet), position)>
```

Può succedere, inoltre, che nelle due CUI un interattore astratto sia mappato allo stesso modo in un interattore concreto.
In questo caso, la differenziazione è garantita per mezzo di differenti valori degli attributi per lo stesso oggetto concreto. Per esempio, se consideriamo un elemento *text_edit* questo può essere implementato con un *textfield* in entrambi i casi.

```
 <!ELEMENT text_edit (textfield)>

<!ELEMENT textfield EMPTY>
<!ATTLIST textfield
      label CDATA #REQUIRED
      length NMTOKEN #REQUIRED
      password (%option;) #REQUIRED>
```

La memorizzazione delle informazioni relative agli altri elementi delle CUI avviene per mezzo di procedimenti analoghi.

3.2.3 *Interfaccia Utente Finale*

L'*Interfaccia Utente Finale* è l'interfaccia utente codificata nel linguaggio di implementazione scelto (Java, XHTML, VoiceXml).
In questo lavoro è stato preso in considerazione l'XHTML.

3.3 Semantic Redesign

In questo paragrafo viene fornita una descrizione dell'algoritmo di *Semantic Redesign* supportato da TERESA [4, 5, 6].

Il processo di trasformazione è attuato attraverso l'uso di due descrizioni logiche (astratta e concreta) della pagina web desktop che permettono di identificare la semantica dell'interazione supportata dalla pagina, per poi individuare un'implementazione adatta alle risorse del dispositivo mobile destinatario.

La descrizione astratta (basata sulla classificazione semantica di interattori e operatori di composizione) è importante perché identifica gli obiettivi di comunicazione originari del *web designer* che dovrebbero essere preservati anche nelle presentazioni per *mobile*[11] create.

La descrizione concreta è inoltre importante perché con l'informazione che fornisce (ad esempio le dimensioni in pixel e l'indicazione delle tecniche di implementazione) permette di valutare più precisamente quanti interattori potrebbero essere inseriti nelle nuove presentazioni o che tipo di interattore potrebbe essere inserito. Ad esempio, un interattore *single selection* può essere rappresentato come un *list box* in una presentazione desktop, ma non può esserlo in una per *mobile* (nel caso in cui il numero di scelte risulta elevato), di conseguenza l'interattore deve essere trasformato in un *drop down list*.

La figura 3.4 mostra la home page di un generico sito web.

[11] Si definisce presentazione per *mobile* una presentazione realizzata per terminali mobili.

Figura 3.4: Tipico layout di una generica pagina web

Dando uno sguardo alla figura 3.4 emerge chiaramente che la pagina web non può essere visualizzata sul display di un dispositivo mobile senza effettuare delle opportune modifiche su di essa.

Le ragioni sono varie.

Innanzitutto la pagina è troppo grande per il piccolo display e contiene delle immagini che al fine di essere visualizzate sul dispositivo mobile devono essere ridotte.

C'è poi il problema del testo: questo, essendo piuttosto lungo, richiede la riduzione dei font del testo ed una suddivisione su più pagine.

Non meno problematiche sono le sezioni menu e link che per essere visualizzati possono essere disposti su un'unica colonna o suddivisi su più pagine.

L'algoritmo di *Semantic Redesign* opera alcune di queste trasformazioni: questo, in particolare, elabora la descrizione logica (CUI-desktop) della pagina web desktop, creata attraverso il processo di *reverse engineering*, e genera una descrizione logica (CUI-mobile)

per il dispositivo mobile, dalla quale è possibile automaticamente ottenere la corrispondente interfaccia utente, mantenendo le caratteristiche semantiche della pagina originaria, attraverso un processo di *forward engineering* (figura 3.5).

L'algoritmo di *Semantic Redesign* decide, inoltre, come gli elementi della pagina debbano essere implementati nel dispositivo mobile destinatario.

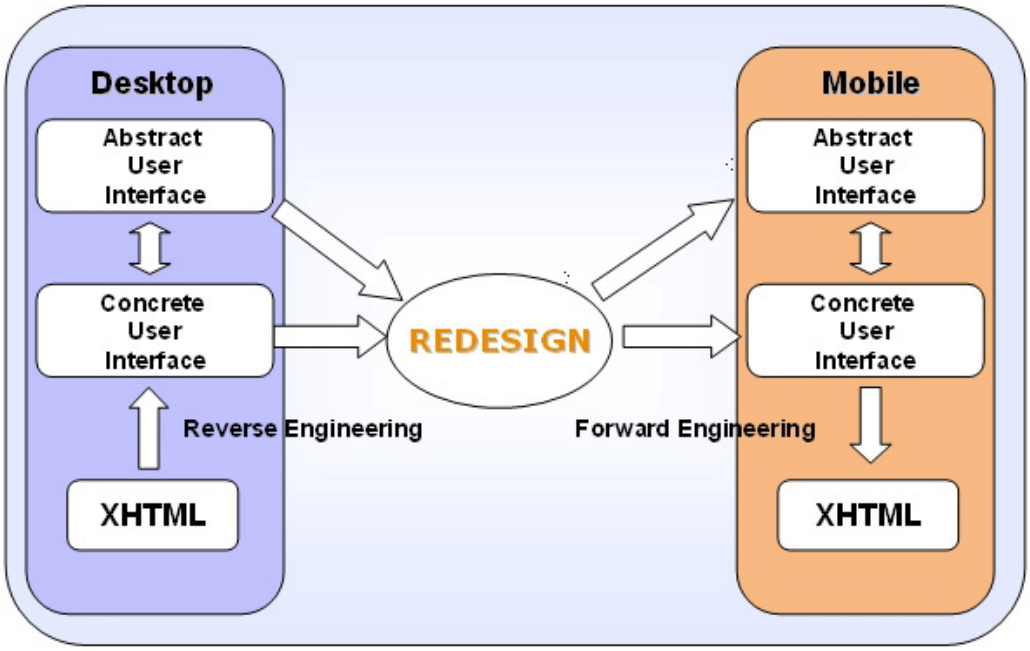


Figura 3.5: Reverse e Forward Engineering nel processo di trasformazione

3.4 Regole applicate nella trasformazione

L'algoritmo di *Semantic Redesign* applica una serie di regole di trasformazione [1] alla descrizione logica concreta (CUI-desktop) per creare la descrizione logica per il dispositivo mobile (CUI-mobile).
Le regole di trasformazione possono essere raggruppate in tre categorie:

1. *Regole di page splitting*: sono applicate all'interfaccia utente dell'applicazione desktop per "splittarla" in parti che sono logicamente e semanticamente correlate.
2. *Regole di trasformazione degli interattori*: sono applicate per trasformare gli elementi dell'interfaccia utente originaria in elementi diversi ma che supportano le stesse funzionalità.
3. *Regole di resizing*: sono usate per ridimensionare elementi e immagini.

Regole di *page splitting*

Vengono ora presentate le regole di *page splitting* che sono alla base dell'algoritmo di *Semantic Redesign*.
In dettaglio, la trasformazione segue questi criteri principali:

- Durante il processo di trasformazione si prendono in considerazione gli operatori di composizione che indicano le relazioni semantiche tra gli elementi che dovrebbero essere preservate nell'interfaccia utente risultante.
- Quando gli interattori di cui un "operatore di composizione" è costituito non possono essere contenuti in un'unica presentazione per *mobile*, vengono suddivisi in più presentazioni collegate da un numero sufficiente di connessioni (ad esempio, link *next* e *prev*).
- Ogni volta che la visita degli interattori di un "operatore di composizione" comporta il loro spostamento in una nuova presentazione, viene creata una connessione tramite un link alla presentazione appena creata.

- Viene creata una connessione ogni volta che si devono collegare le nuove presentazioni per permettere la navigazione inversa (tramite, ad esempio, un link *prev*).

Regole di trasformazione degli interattori

Queste regole permettono di cambiare l'implementazione degli interattori in relazione alle risorse d'interazione disponibili nel dispositivo mobile.
Qui di seguito vengono presentate le regole di trasformazione a cui sono soggetti alcuni elementi (*select*, *radio button*, *textarea*) di una form XHTML passando dalla piattaforma desktop a *mobile*.

Tag <select>

Trasformazione degli elementi *single select* e *multiple select* dalla piattaforma desktop a quella *mobile* in base alla cardinalità:

- *single select*
  - cardinalità bassa (<=3) → radio button
  - cardinalità alta (>3) → drop down list
  - cardinalità molto alta (>10) → textfield
- *multiple select*
  - qualsiasi cardinalià → check box

Tag <radio button>

Trasformazione dell'elemento *radio button* dalla piattaforma desktop a quella *mobile* in base alla cardinalità:

- *radio button*
  - cardinalità bassa (<=3) → radio button
  - cardinalità medio-alta (>3) → drop down list

Esempio di trasformazione dell'elemento *radio button* con cardinalità medio-alta (figura 3.7).

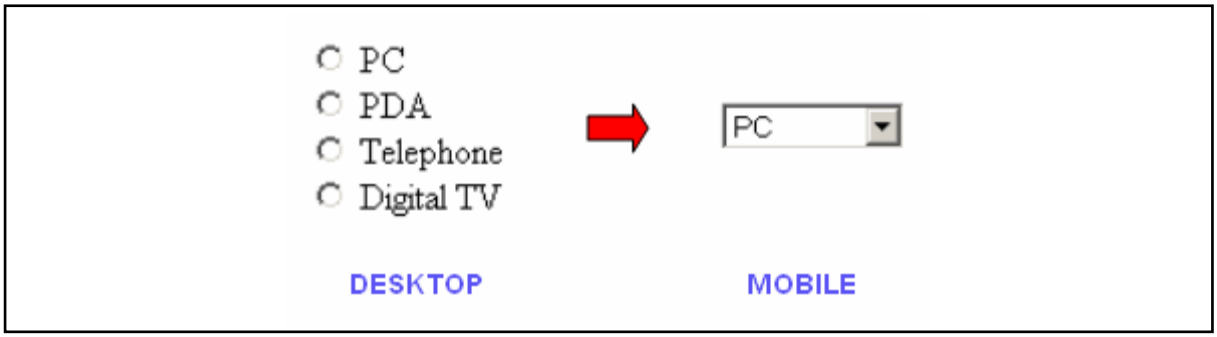


Figura 3.7: Esempio di trasformazione dell'elemento *radio button*

Tag <textarea>

Trasformazione dell'elemento *textarea* dalla piattaforma desktop a quella *mobile* (figura 3.8):

- textarea → textfield

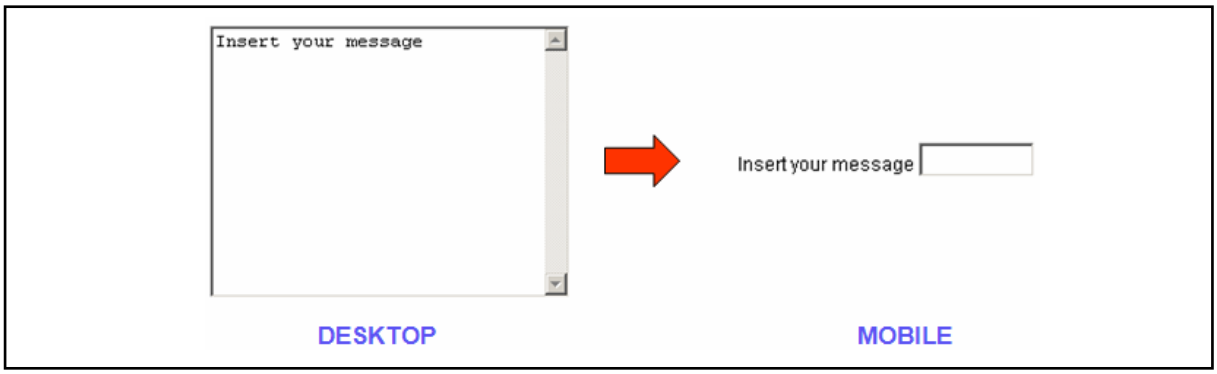


Figura 3.8: Esempio di trasformazione dell'elemento *textarea*

Regole di *resizing*

Qualora accada che la dimensione di un'immagine contenuta in una pagina web sia più larga del display del dispositivo mobile che deve visualizzarla, questa deve essere ridotta per poterla adattare alle risorse del dispositivo, eseguendo un algoritmo di *resize* dell'immagine.

Le immagini vengono ridimensionate in una risoluzione orizzontale massima di 150 pixel mantenendo la stessa *aspect-ratio*.

La figura 4.1 mostra un esempio di trasformazione di una immagine.

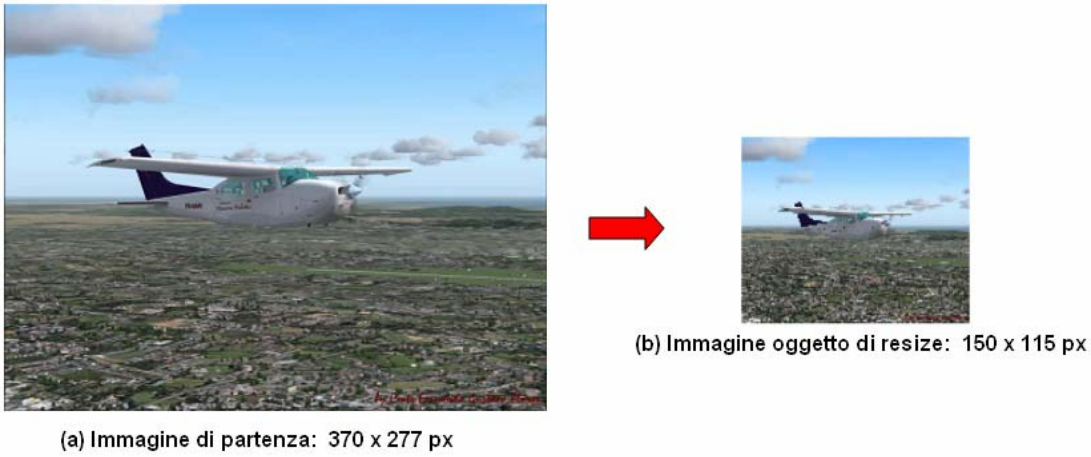


Figura 4.1: Esempio di trasformazione di una immagine

3.5 Dettagli progettuali

Viene ora descritto in dettaglio il processo di trasformazione della CUI-desktop nella CUI-mobile svolto dall'algoritmo di *Semantic Redesign*.

La CUI-desktop è costituita da un solo elemento `presentation` poiché l'algoritmo analizza una pagina web per volta.

Ogni elemento `presentation` è caratterizzato da una struttura che descrive l'organizzazione statica dell'interfaccia utente (elementi `interactor` e `interactor_composition`) e da zero o più elementi `connection` che forniscono informazioni circa le relazioni tra gli elementi della presentazione.

Il frammento di codice XHTML mostra due elementi: `<p>` e `<a>` racchiusi da un `<div>`.

```
<div>
     <p>text</p>
     <a href=”page_dest.html”>Go</a>
</div>
```

Il codice riportato di seguito rappresenta una parte della CUI-desktop corrispondente:

```
<?xml version="1.0" encoding="ISO-8859-9"?>
<!DOCTYPE concrete_desktop_interface PUBLIC
      "HIIS laboratory DTD of Desktop CUI" "Concrete-UI-Desktop.dtd">
<concrete_desktop_interface>
    <default_settings>
        [ … ]
    </default_settings>

    <presentation name="PRES_1">
          <presentation_properties>
                [ … ]
          </presentation_properties>

          <connection presentation_name="PRES_2">
                 [ … ]
          </connection>

          <interactor_composition>

                <operator id="Grouping">
                […]
                </operator>

                <first_expression>
                      <interactor id="Text">
                      [ …]
                      </interactor>
                </first_expression>

                <first_expression>
                      <interactor id="Link">
                       [ …]
                      </interactor>
                </first_expression>

          </interactor_composition>

    </presentation>
</concrete_desktop_interface>
```

CUI-desktop.xml

Tralasciando i `default_settings` generali in cima al file e le `presentation_properties` della presentazione, possiamo notare che il file contiene una sola presentazione (PRES_1).

Possiamo inoltre constatare che la presentazione PRES_1 è composta da un elemento `interaction_composition` contenente al suo interno due elementi `interactor` (Text e Link).

Vediamo allora in dettaglio come è strutturata la presentazione PRES_1:

- contiene un nodo complesso radice G_0 (`interactor_composition`) il cui operatore *grouping* raggruppa 2 `interactor` (Text e Link).

La situazione è rappresentata in figura 3.9:

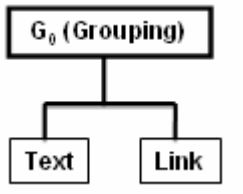


Figura 3.9: Struttura dell'albero DOM della CUI-desktop per il frammento di codice

Alcune parti della DTD-desktop corrispondente:

```
<!ELEMENT presentation (presentation_properties,
                        connection*,
                        (interactor | interactor_composition))>

<!ELEMENT interactor (interaction | only_output)>

<!ELEMENT interactor_composition (operator,
                                  first_expression+,
                                  second_expression?)>

<!ELEMENT operator (grouping | ordering | hierarchy | relation)>

<!ELEMENT first_expression (interactor | interactor_composition)>

<!ELEMENT second_expression (interactor | interactor_composition)>
[……]
```

La creazione della CUI-mobile a partire dalla CUI-desktop si articola in tre fasi (figura 3.5): *ParsingCUI, SplittingCUI, TransformCUI, GeneratorCUI*.

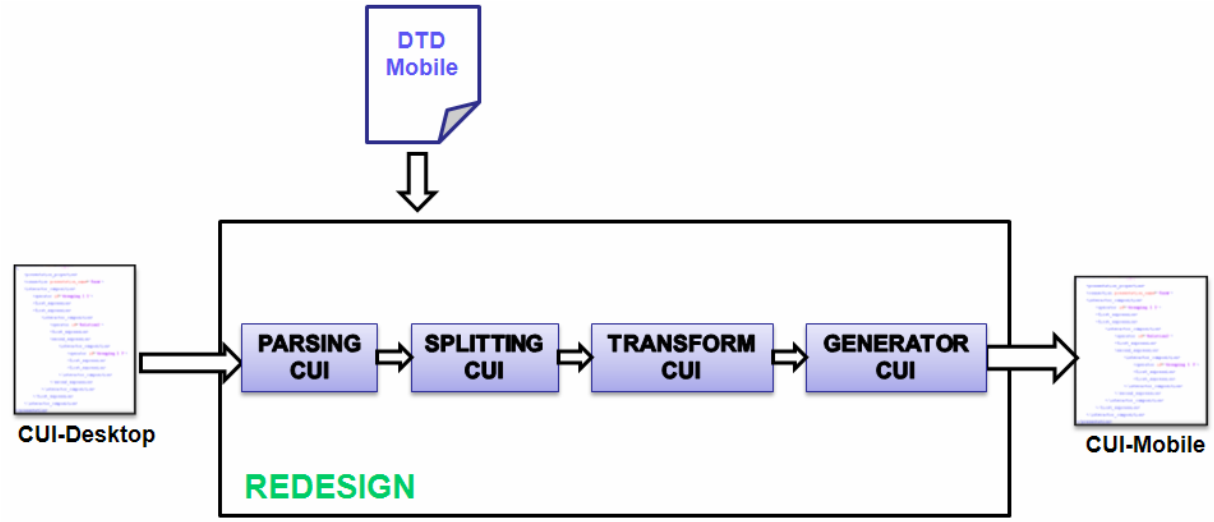


Figura 3.5: Fasi dell'algoritmo di Semantic Redesign

ParsingCUI

Il processo di trasformazione inizia dalla specifica XML della Concrete User Interface (CUI-desktop).

Per poter analizzare la CUI-desktop al fine di trasformarla, questa passa attraverso un parser HTML che crea l'albero DOM della stessa.

La CUI-desktop contiene elementi `interactor` (corrispondono agli interattori: text, image, text_edit, single_choise, multiple_choise, ecc) ed elementi `interactor_composition` (corrispondono agli operatori di composizione: Grouping, Ordering, Hierarchy e Relation).

Un `interactor_composition` può a sua volta essere composto da elementi `interactor` e `interactor_composition` annidati.

Durante questa fase ogni `interactor` viene mappato, a livello implementativo, in un oggetto `SimpleElem`, mentre ogni `interactor_composition` in un oggetto `ComplexElem`.

Il risultato della fase *ParsingCUI* è una struttura ad albero multi-livello (implementata come *hashtable*) i cui nodi interni rappresentano gli operatore di composizione (`interactor_composition`) e le cui foglie rappresentano gli interattori (`interactor`).

SplittingCUI

Durante questa fase viene creato un array di presentazioni (`NewPres[] arrNewPres`) corrispondente alle pagine web per il dispositivo mobile.
L'algoritmo crea la prima presentazione per *mobile* che contiene il nodo radice e chiama su di esso una funzione ricorsiva `dividiPag2()`.
La funzione effettua una visita *Depth First Search* (DFS) dell'albero. Se durante la visita del DOM incontra un `SimpleElem` lo inserisce nella presentazione corrente, mentre, se incontra un `ComplexElem`, ne crea una nuova, inserisce un link nella presentazione corrente e chiama ricorsivamente la funzione su quel nodo.
Quando si raggiunge il numero di 6 interattori inseriti nella presentazione, l'algoritmo ne crea una nuova e procede inserendo gli interattori.
In questa fase l'algoritmo tiene conto delle regole di *page splitting* discusse nel paragrafo precedente.
Alla fine ogni presentazione per *mobile* conterrà un array di connessioni e un array di interattori e operatori di composizione.

TransformCUI

Durante questa fase l'algoritmo applica le regole di trasformazione degli interattori per rendere la CUI-mobile conforme alla DTD-mobile.

GeneratorCUI

Nella fase *GeneratorCUI* a partire dall'array delle presentazioni, contenente gli oggetti `SimpleElem`, `ComplexElem` e le connessioni, viene scritta la CUI-mobile su file, cioè viene mappato ogni oggetto nel corrispondente codice XML equivalente.

Capitolo 4

Cost-based Semantic Redesign

4.1 Introduzione

In questo capitolo viene presentata una nuova versione dell'algoritmo di *Semantic Redesign* basata sul "costo" che estende l'algoritmo pre-esistente, applicabile solo in determinati casi dal momento che presentava delle limitazioni molto rigide:

- L'algoritmo divideva la pagina desktop originale ogni 6 interattori (elementi `interactor`) senza considerare di che tipo fossero o quante risorse occupassero nella pagina per *mobile*.
- Gli interattori raggruppati da un operatore di composizione (elemento `interactor_composition`), venivano sempre spostati in una nuova presentazione e sostituiti da un link alla nuova presentazione.

4.1.1 Nuovo approccio basato sul costo

Nel nuovo processo di trasformazione delle pagine web vengono presi in considerazione non solo gli aspetti semantici, ma anche il costo in termini di spazio occupato dagli elementi che compongono l'interfaccia utente (testo, immagini, bottoni, ecc).
Nella versione precedente era stato fissato un numero massimo di interattori che potevano far parte di una presentazione per *mobile*.
Questo però risultava essere troppo rigido, tanto che si è deciso di introdurre un costo in base al quale definire il numero massimo di interattori da inserire nelle presentazioni.

In questo modo ogni interattore e ogni operatore di composizione ha associato un costo differente che dipende da alcuni fattori.

Per costo di un elemento XHTML (se consideriamo la pagina web) o interattore/operatore di composizione (se consideriamo la CUI-desktop e CUI-mobile) si intende lo spazio che l'elemento/interattore/operatore di composizione occupa nel display del terminale mobile. Questo è definito in unità del display, cioè pixel.

L'algoritmo inserisce gli interattori in una presentazione per *mobile* finché la somma del costo di ogni interattore e operatore di composizione non raggiunge il massimo costo supportato dalla presentazione.

I fattori che determinano il costo degli interattori sono:

- la dimensione del font dei caratteri (in pixel)
- il numero di caratteri in un testo (lunghezza del testo)
- la dimensione delle immagini (in pixel)
- il valore dell'interspazio tra un interattore e l'altro (in pixel)
- il tipo di interattore (testo, immagine, link, ecc) e operatore di composizione (Grouping, Relation, Ordering, Hierarchy).

4.2 Nuove regole applicate nella trasformazione

Nel presente paragrafo vengono descritte le regole introdotte nel nuovo algoritmo per effettuare il processo di trasformazione delle pagine.
Queste possono essere raggruppate come segue:

- Regole di *page splitting*
- Regola di trasformazione dell'interattore "testo"
- Regola di navigazione delle pagine web

Regole di *page splitting*

Qui di seguito vengono elencate le regole illustrare nel capitolo 3 che sono state oggetto di modifica:

- Il numero di interattori che una pagina per *mobile* può supportare non è più fisso, ma variabile e dipendente dal costo.
- Gli interattori raggruppati da un operatore di composizione (elemento `interactor_composition`) sono collocati in una nuova presentazione per *mobile* e sostituiti da un link alla nuova presentazione, solo se si supera il costo massimo supportato dalla presentazione.
- Se il numero di interattori raggruppati da un operatore di composizione è elevato, questi vengono distribuiti su più presentazioni tenendo conto del costo che ciascuna presentazione può supportare.

Regola di trasformazione dell'interattore "testo"

In presenza di testi lunghi viene applicata una nuova regola per minimizzare l'attività di scrolling dell'utente.

In particolare se il numero di caratteri del testo supera il valore della soglia che è stata definita (`LIMITE_TESTO_LUNGO`), il testo viene suddiviso in pezzi più corti visualizzabili su più pagine sul dispositivo mobile.

Le pagine ottenute conterranno tutte la stessa lunghezza di testo ad eccezione della prima pagina, che contiene soltanto la prima riga di testo e un link al restante.

La figura 4.2 mostra un esempio di splitting di un testo lungo.

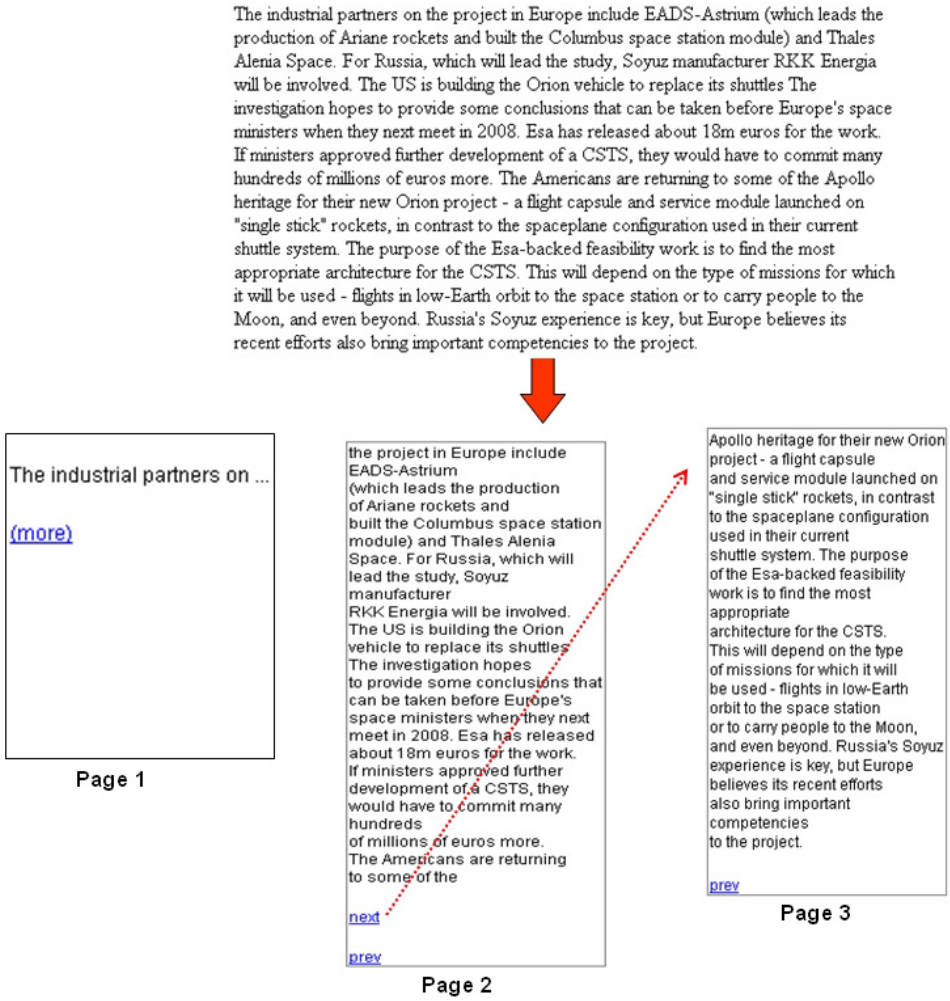


Figura 4.2: Esempio di splitting di un testo lungo

Regola di navigazione delle pagine web

In generale, esistono due approcci tradizionali per navigare le pagine web nei dispositivi mobili:

- *scrolling*
- *page splitting*

Con il primo approccio, l'utente si trova a dover continuamente effettuare lo scrolling su/giù o sinistra/destra per vedere il contenuto della pagina web, operazione noiosa e difficile quando vuole localizzare un'informazione precisa all'interno di essa.
Nel secondo approccio, invece, una pagina web è suddivisa in pagine di dimensioni ridotte adattate al display del dispositivo mobile ed è possibile quindi che l'utente, se il numero di pagine è elevato, perda l'orientamento mentre naviga.
La soluzione proposta è una via di mezzo tra l'approccio del *page splitting* e l'approccio dello *scrolling*.
Lo scrolling orizzontale è eliminato del tutto, mentre quello verticale è ridotto.

4.3 Dettagli progettuali

In questo paragrafo viene presentata la parte centrale del modulo *Redesign* che esegue la trasformazione della CUI-desktop per ottenere la CUI-mobile.

Il processo di trasformazione è guidato dall'algoritmo di *Cost-based Semantic Redesign* che si basa su diverse regole di trasformazione.

Fondamentalmente la costruzione della CUI-mobile avviene attraverso varie fasi in sequenza (figura 4.3): *ParsingCUI*, *TransformCUI*, *CalcolateCost*, *SplittingCUI*, *GeneratorCUI*.

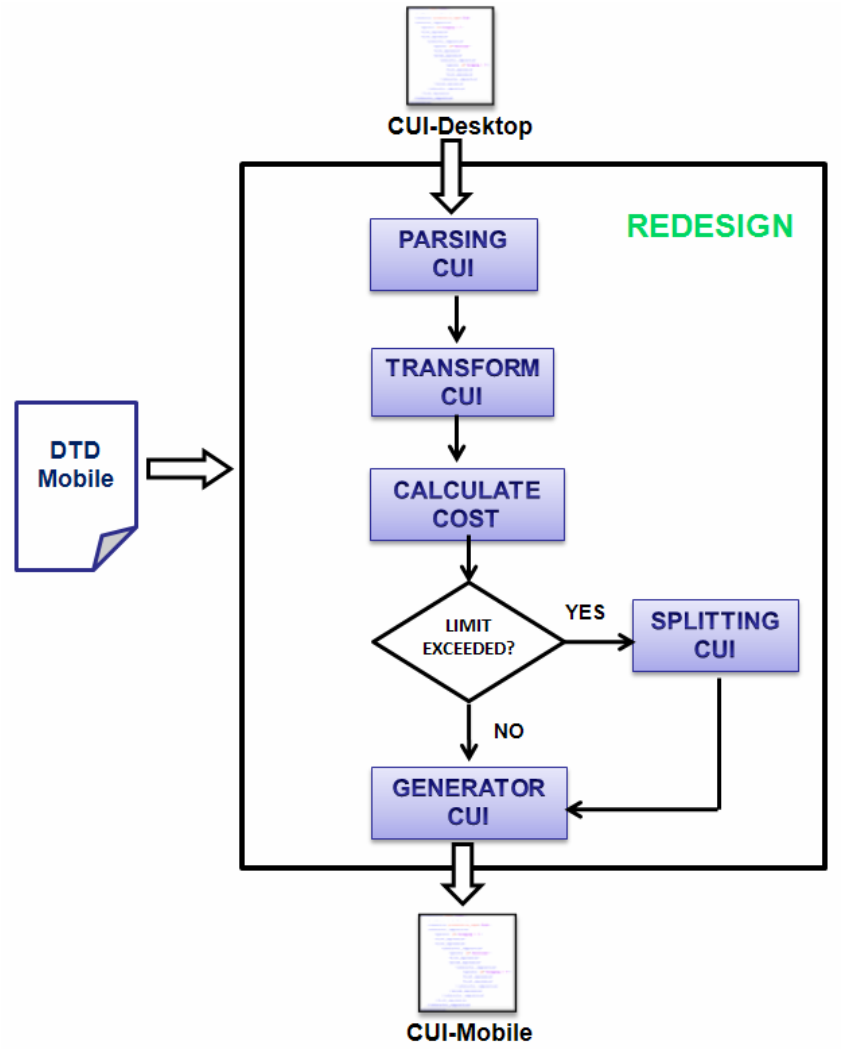


Figura 4.3: Fasi dell'algoritmo di *Cost-based Semantic Redesign*

Il modulo *Redesign* utilizza la stessa struttura dati creata nella fase di *ParsingCUI* discussa nel capitolo 3 e basandosi sul DOM della CUI-desktop crea una struttura ad albero multi-livello (implementata come *hashtable*) le cui foglie contengono gli interattori, i nodi intermedi gli operatori di composizione e la radice l'intera pagina.

4.3.1 TransformCUI

Durante questa fase l'algoritmo applica le regole di trasformazione agli interattori della CUI-desktop per far si che il costo venga calcolato sull'interattore oggetto di trasformazione.
Le regole applicate sono:

- Trasformazione dei testi lunghi
- Trasformazione delle immagini
- Trasformazione di altri oggetti dell'interfaccia (radio button, textarea, drop-down list)

Esempio: un *radio button* con cardinalità > 3 viene trasformato nel dispositivo mobile in un *drop-down list*.
Così nella fase successiva, il costo sarà calcolato direttamente sull'interattore trasformato (*drop-down list*).

4.3.2 CalculateCost

Nella fase di *CalculateCost* viene eseguita la procedura `CalculateCost()` per calcolare il costo di ogni `SimpleElem` e `ComplexElem` memorizzati nella struttura ad albero. L'algoritmo può essere descritto con la seguente procedura ricorsiva.

PSEUDO CODICE

```
   PROCEDURA
1. CalculateCost(ComplexElem root)
2. {
3.   FOR each figlio E di root
4.   {
5.       IF E è un SimpleElem
6.       {
7.          IF(E.type == testo)
8.                /* calcola il costo del testo  */
9.          IF(E.type == immagine)
10.               /* calcola il costo dell'immagine  */
11.         IF(E.type == link)
12.               /* calcola il costo del link */
13.         IF(E.type == radio_button)
14.               /* calcola il costo del radio button */
15.         IF(E.type == checkbox)
16.               /* calcola il costo del checkbox */
17.         IF(E.type == select)
18.               /* calcola il costo del select */
19.         IF(E.type == interactive_description)
10.            /* calcola il costo dell'interactive_description*/
21.         [……]
22.         Setta il campo costo del SimpleElem
23.       }
24.       ELSE {
25.           /*  è un ComplexElem  */
26.           CalculateCost(E)
27.       }
28.   }
29.   costo del ComplexElem è definito come la somma dei costi
       dei figli + eventuale costo della "cornice".
30.}
```

Quando viene eseguita la procedura `CalculateCost()`con il nodo radice come parametro, l'algoritmo esegue una visita in profondità (DFS) dell'albero: se incontra un `SimpleElem` calcola il costo direttamente (cioè, in base al tipo di interattore, viene eseguita una funzione corrispondente che ne calcola il costo), mentre se incontra un `ComplexElem` applica ricorsivamente la procedura su quel nodo.

Il costo di un `ComplexElem` è definito come la somma dei costi dei suoi figli più, se presente, il costo di una "cornice" in presenza di particolari elementi (es. `fieldset`).

Alla fine di questa fase, ad ogni `SimpleElem` e `ComplexElem` sarà assegnato il proprio costo.

La procedura `CalculateCost()` calcola il costo dei seguenti interattori:

- Testo
- Immagini
- Link
- Elementi di una Form:
    - Texfield
    - Reset e submit button
    - Radio button, checkbox
    - Select
- Elemento Interactive_description (testo e link annidati)

Di seguito sono descritti alcuni dei metodi usati per calcolare il costo degli interattori.

Calcolo del costo dell'interattore "testo"

Il frammento di codice seguente mostra la chiamata del metodo `getCostTEXT()` che si occupa del calcolo del costo del testo.

```
[ … ]

 // identifica l'interattore testo
 if(textBegin != -1)
 {
   // recupera il testo effettivo dall'interattore
   String text = textNode.substring(textBegin + 18, end);

   /* 300 caratteri esclusi gli spazi è il limite
         oltre il quale si considera "testo lungo" */
   if(text.length() > LIMITE_TESTO_LUNGO)
   {
      // setta un campo dell'oggetto per indicare che è un "testo lungo"
      se.set_type("testolungo");
   }

   se.set_testo(text);

   // chiama il metodo che calcola il costo del testo
   cost = getCostTEXT(text);
}

[ … ]
```

Metodo: getCostTEXT()

Il metodo `getCostTEXT(String text)` prende come parametro il testo effettivo dell'interattore e calcola il numero di caratteri di cui è composto.

In particolare se quest'ultimo è maggiore del valore della soglia che è stata definita `LIMITE_TESTO_LUNGO`, ad esempio 300 caratteri, allora assegna alla variabile `cost` la costante `COSTO_SOSTITUZIONE_TESTO_LUNGO` che rappresenta il costo dell'interattore in cui verrà trasformato (cioè una linea di testo, uno spazio bianco e un link). Se al contrario non siamo in presenza di un testo lungo si calcola il costo del testo normalmente (cioè numero di linee moltiplicato per il costo di una linea di testo) avendo supposto che la linea di testo che il terminale mobile riesce a visualizzare è di 32 caratteri.

```
// calcola il costo del testo
public static int getCostTEXT(String text)
{
      int cost = 0;

      // numero di caratteri
      int characters = text.length();

      if(characters > LIMITE_TESTO_LUNGO)
      {
            // costo una linea di testo, uno spazio bianco e un link
            cost = COSTO_SOSTITUZIONE_TESTO_LUNGO;

      } else {
            /* Calcola il numero di linee e
               lo moltiplica per il costo di una linea di testo  */
      cost = ( ( ((int) characters /32) + 1)*15 + 20 );
      }

      return cost;
}
```

Esempio di trasformazione del testo lungo

Dal momento che il testo lungo (numero di caratteri > 300) viene sicuramente trasformato nel dispositivo mobile in una *linea di testo* + *uno spazio bianco* + *un link*, il costo che viene assegnato all'interattore è dato dalla somma del costo della linea di testo, dello spazio bianco e del link .

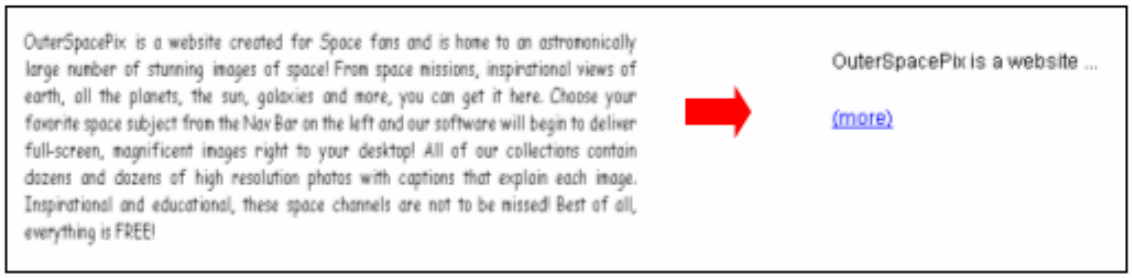


Figura 4.4: Trasformazione di un testo lungo da desktop a mobile

Calcolo del costo dell'interattore "immagine"

Il frammento di codice seguente mostra la chiamata del metodo `getCostIMG()` che calcola il costo di un'immagine.

```
[ … ]

// identifica l'interattore immagine
if(img != -1)
{
  // il resize dipende dal tipo di dispositivo(big, medium, small)
  String device = "big";

  // recupera il path dell'immagine
  String imgsrc = ……;

  // esegue il metodo di resize che restituisce le nuove dimensioni
  int[] WeH = ResizeImage(imgsrc, device);

  // chiama la funzione che calcola il costo
  cost = getCostIMG(WeH);

}
```

Prima di calcolare il costo, viene eseguito il metodo `ResizeImage(…)` che ridimensiona l'immagine e restituisce i valori ridotti della dimensione orizzontale e verticale dell'immagine.

Metodo: getCostIMG()

Il metodo `getCostIMG(int[] WeH)` prende come parametro un array di 2 elementi che corrispondono uno alla dimensione orizzontale e l'altro a quella verticale dell'immagine che è stata sottoposta a *resize*.

Il costo è dato dalla dimensione verticale dell'immagine.

```
// calcola il costo dell'immagine
public static int getCostIMG(int[] WeH)
{
      int cost = 0

      if( (WeH[1] > 1) && (WeH[1] <= 150) )
      {
        // il costo è dato dalla dimensione verticale dell'immagine
        cost = WeH[1];
      }

      return costo;
}
```

Calcolo del costo dell'interattore "radio button" di una form

Il frammento di codice seguente mostra la chiamata del metodo `getCostRADIO_BUTTON()` che calcola il costo di un elemento *radio button* di una form.

```
[…]
// elemento radio button di una form
if(radio != -1)
{
      // recupera il codice XML relativo all'interattore
      String text = testoNodo;

      int low_card = text.indexOf("low_card");
      int medium_card = text.indexOf("medium_card");

      // determina la cardinalità dell'interattore
      if(low_card != -1)
      {
            cost = getCostRADIO_BUTTON(text, "low_card");
      }
      else if (medium_card != -1)
           {
                 cost = getCostRADIO_BUTTON(text, "medium_card");
           }
}
[…]
```

Metodo: getCostRADIO_BUTTON()

Il metodo `getCostRADIO_BUTTON()` prende come parametri il codice XML corrispondente all'interattore ed il valore della cardinalità dell'elemento.
In base a quest'ultimo si calcola poi il costo dell'interattore.

```
// calcola il costo di un radio button
public static int getCostRADIO_BUTTON(String text, String cardinality)
{
      int cost = 0;

      if(cardinality.equals("low_card"))
      {
            int numChoise = tmp.indexOf("choice_element");
            // conta il numero delle occorrenze di "choice_element"
            int num = 0;

            // determina il numero di "choice_element"
            if(numChoise != -1)
            {
                  StringTokenizer st = new StringTokenizer(tmp, " <>");

                  while (st.hasMoreTokens())
                  {
                        String val = st.nextToken();
                        if(val.equals("choice_element"))
                        {
                              num++;
                        }
                  }
            }

         //Costo = costo del singolo elemento * #elementi + spazio bianco
            cost = costoElementoRadio * numElementi + costoSpazio;

      } else if(cardinality.equals("medium_card"))
               {
                  // costo dell'elemento "Select"
                  cost = costoElementoSelect + costoSpazio;
               }
      return cost;
}
```

Esempio

Dal momento che un *radio button* con cardinalità "medio-alta" viene trasformato nel dispositivo mobile in un *select* (figura 4.5), allora il costo che si assegna all'interattore è quello del *select*.

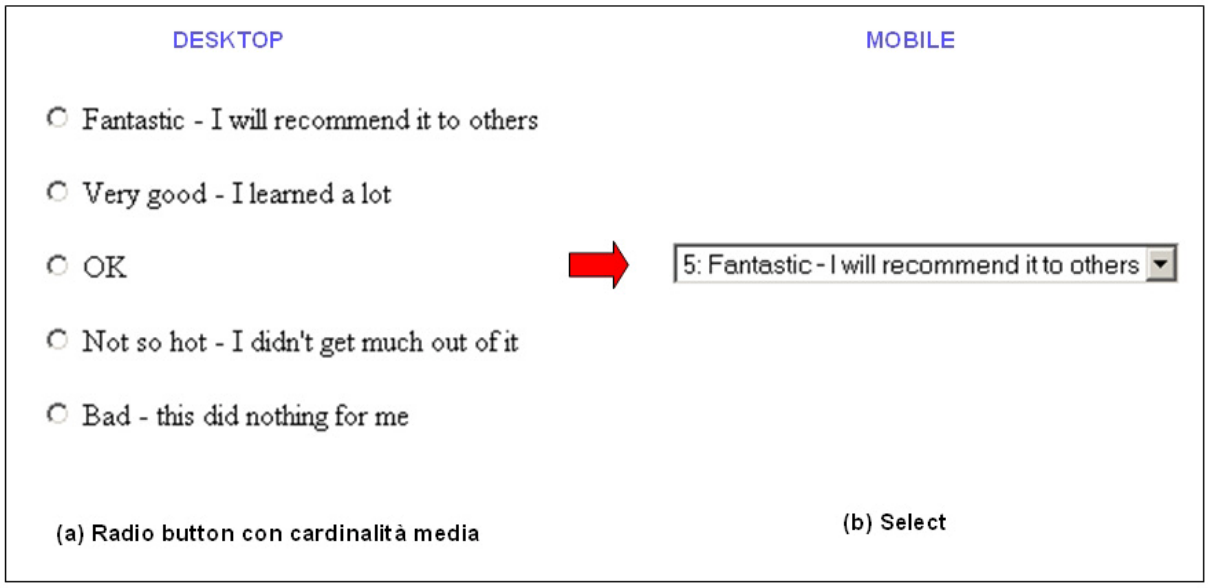


Figura 4.5: Trasformazione di un "radio button con cardinalità media" da desktop a mobile

4.3.3 SplittingCUI

Durante la fase di *SplittingCUI* si decide come devono essere disposti gli interattori nelle presentazioni per mobile tenendo conto degli operatori di composizione.

L'operazione di *splitting* è svolta attraverso una procedura ricorsiva applicata al `ComplexElem` radice.

PSEUDO CODICE

```
   PROCEDURA
1. PAGE_SPLITTING (ComplexElem root) {
 //determina i figli ComplexElem di root che possono essere ridotti a link
2.   calcola il costo di root
3.   WHILE(il costo di root > COSTO_MAX_PAGE_MOBILE) {
4.      N = figlio ComplexElem di root di costo massimo
5.      IF(N != NULL) {
6.        aggiorna il suo costo come link
7.        setta un bool a TRUE per indicare che deve essere ridotto a link
8.      } ELSE {
9.           /* Esce dal while  */
10.       }
11.   }
12.   crea la Presentazione P
      //Inserisce gli elementi nelle Presentazioni
13.   FOR ogni elemento E dei figli di root {
14.      IF E è un SimpleElem {
           //I SimpleElem possono essere distribuiti su più presentazioni
15.         IF(costo(P) < COSTO_MAX_PRES){
16.             inserisce E in P
17.         } else {
18.           Crea una nuova Presentazione P
19.           Crea un link a P
20.           Inserisce E in P
21.         }
22.      } ELSE {
23.            /*  è un ComplexElem  */
24.            IF(E è ridotto a link) {
25.               inserisce un link a E in P
26.               chiama PAGE_SPLITTING(E)
27.            } ELSE {
28.                 inserisce E in P
29.              }
30.        }
31.   }
32.}
```

La procedura di *page splitting* prevede due sotto-fasi. Durante la prima fase vengono per prima cosa determinati i figli del `ComplexElem` in esame che possono essere ridotti a link. Poi viene eseguito il metodo `reduceAtLink(…)` che effettua un ciclo fino a che il costo del `ComplexElem` in esame non raggiunge la soglia che è stata definita `COSTO_MAX_PAGE_MOBILE`. Ad ogni ciclo si determina il `ComplexElem` figlio che ha costo massimo per sostituirlo con un link: si aggiorna il campo costo dell'oggetto e si setta un booleano a true che servirà nella sotto-fase successiva per capire se inserire il `ComplexElem` per intero o sostituirlo con un link. A questo punto si ricalcola il costo con il nuovo valore.

Nella seconda sotto-fase, vengono inseriti gli elementi nelle presentazioni per *mobile*: se sono dei `SimpleElem` vengono inseriti direttamente nella presentazione, mentre se sono dei `ComplexElem` viene testato il valore del booleano che era stato settato nella sotto-fase precedente e se questo è `true` si inserisce un link al posto del `ComplexElem` e quest'ultimo viene spostato in una nuova presentazione, invece, se risulta `false` si inserire direttamente il `ComplexElem` nella presentazione corrente.

È stato inoltre fissato un costo limite che le presentazioni per *mobile* possono supportare `COSTO_MAX_PRES`, superato il quale gli interattori vengono distribuiti su più presentazioni.

4.3.4 GeneratorCUI

La fase di *GeneratorCUI* ha il compito di creare il file XML della CUI-mobile e consiste in particolare nell'esecuzione della procedura

```
scriviMobilePages(NewPres[] NewPresArray, Vector cnt)
```

che scandisce l'array delle presentazioni ottenuto nella fase di *SplittingCUI* per creare, per ogni componente della presentazione, il corrispondente codice XML.

Esempio:

- Presentazione → tag <presentation>
- Connessione → tag <connection>
- SimpleElem → tag <interactor>
- ComplexElem → tag <interactor composition>

Nel creare gli interattori tiene conto delle regole di trasformazione per garantire la conformità con la DTD-mobile.

Esempio di trasformazione di un interattore "testo"dalla CUI-desktop alla CUI-mobile

Per garantire che la CUI-mobile sia conforme alla DTD-mobile, il codice dell'interattore della CUI-desktop in fig. 4.6b è soggetto a delle modifiche. Viene copiato tutto tranne quattro attributi (*align, italic, underlined, bold*) che non vengono considerati idonei per la piattaforma mobile (come si può notare nella figura 4.6c).

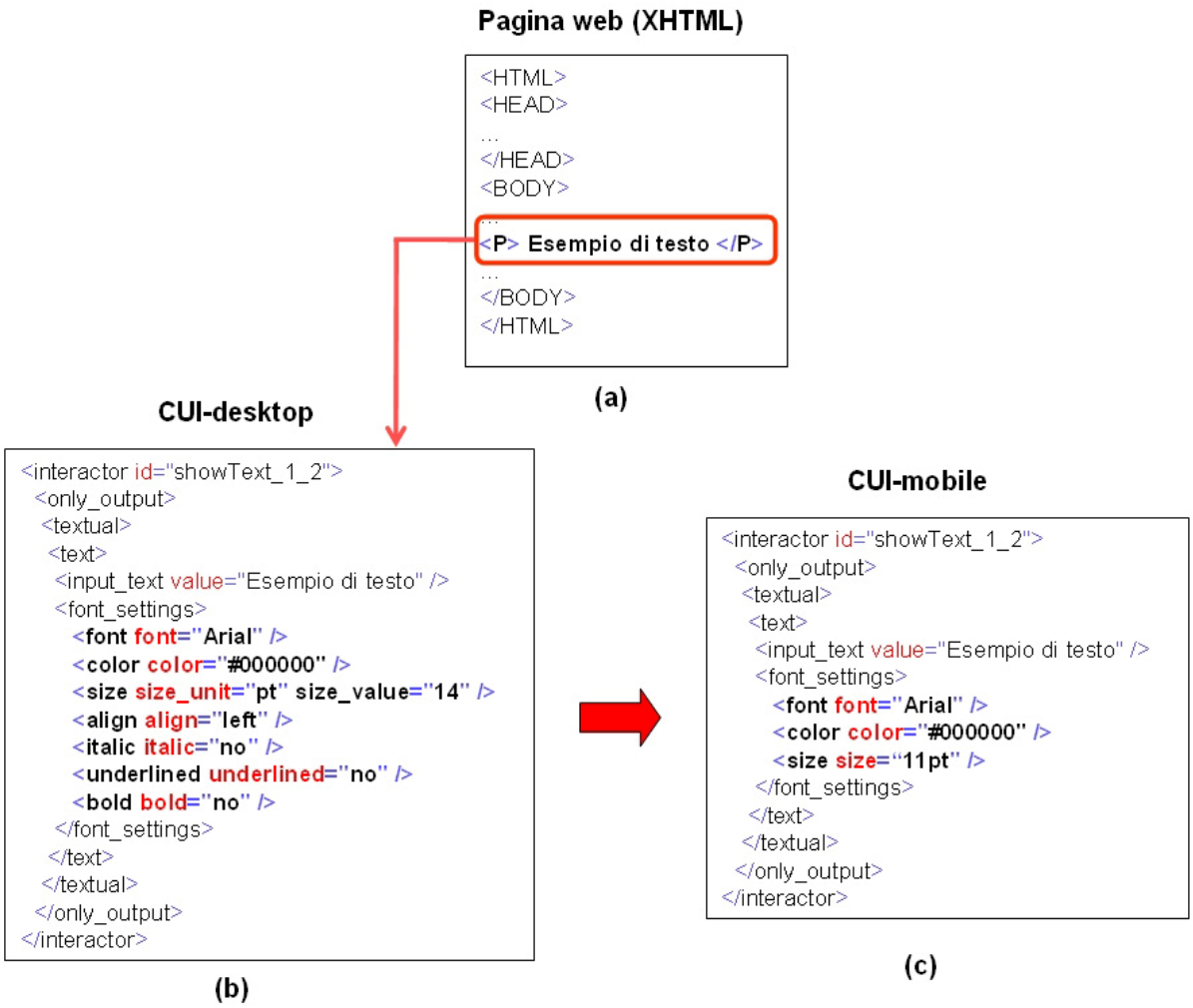


Figura 4.6: Tag XHTML e interattori corrispondenti nelle CUI

Capitolo 5

Il sistema di trasformazione

5.1 Il sistema di trasformazione

L'ultima fase del progetto di tesi è stata dedicata alla realizzazione di un prototipo di sistema basato su un'architettura proxy server [3, 8] che trasforma automaticamente le pagine web, originariamente realizzate per piattaforma desktop, per riadattarle alle risorse dei dispositivi mobili destinatari.

La scelta di implementare il processo di trasformazione delle pagine su un proxy piuttosto che sul browser di un terminale mobile dipende da vari motivi. Uno di questi è che un proxy di solito gira su server con prestazioni più elevate rispetto a un qualunque dispositivo mobile e inoltre può essere condiviso da più dispositivi.

Un altro motivo è che gli utenti dei terminali mobili di solito hanno i propri browser (in dotazione con il terminale mobile) e quindi difficilmente sono disposti a passare a un nuovo tipo di browser o ad installare nuove estensioni.

L'applicazione realizzata è installata su un proxy e, in maniera "trasparente", restituisce le pagine adattate ai terminali mobili che fanno richiesta tramite questo proxy.

Come mostra la figura 5.1, quando un dispositivo mobile richiede una pagina web attraverso il proxy, quest'ultimo recupera la pagina dal server web e la "splitta" per adattarla alle risorse del dispositivo mobile destinatario.

Figura 5.1: Prototipo di sistema basato su architettura proxy server

Qui di seguito verranno descritte l'architettura dettagliata e le componenti principali del prototipo di sistema.

5.2 Architettura dettagliata

Il sistema realizzato è fatto partire come una *Java Servlet* sul server *Apache Tomcat*[12] ed agisce come un proxy tra gli utenti dei terminali mobili e Internet. La sua architettura comprende i seguenti moduli: *Interface Manager*, *Page Manager*, *Reverse*, *Redesign*, *Pages Generator* (figura 5.2).

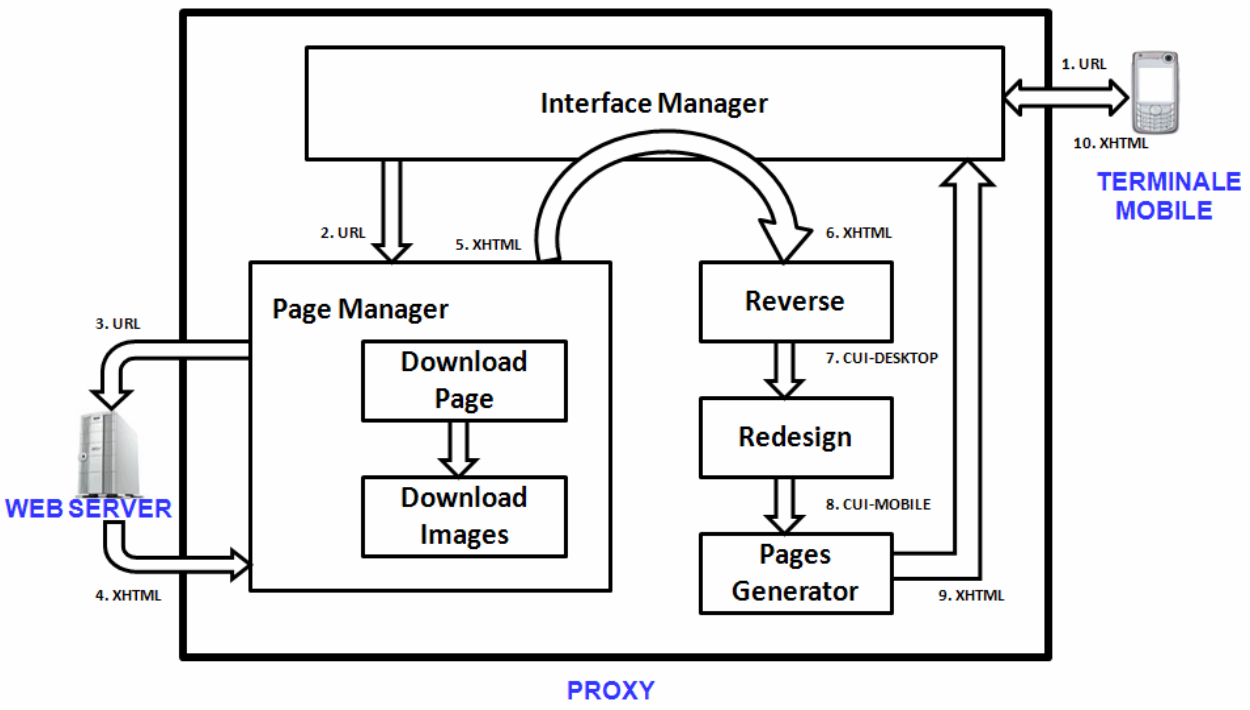


Figura 5.2: Architettura dettagliata del sistema

[12] http://tomcat.apache.org

5.2.1 Modulo *Interface Manager*

Il modulo *Interface Manager* gestisce le richieste HTTP provenienti dai terminali mobili. In particolare, prima di far partire il processo di trasformazione esegue un riconoscimento del dispositivo mobile che sta facendo la richiesta per avere delle informazioni sulle risorse dello stesso (es. la risoluzione del display).

Alla fine del processo di trasformazione della pagina web, il modulo *Interface Manager* restituisce al terminale mobile la prima delle pagine ottenute dalla trasformazione. Se per caso il processo di trasformazione non dovesse andare a buon fine, il modulo ripropone all'utente la pagina iniziale del sistema per poter inserire un nuovo URL.

Profilo del dispositivo mobile

Il modulo *Interface Manager* per riconoscere il dispositivo mobile utilizza la specifica *User Agent Profile* (UAPROF) [30], uno specifico *repository* CC/PP che contiene le descrizioni dei dispositivi mobili.

In generale esistono tre metodi [18]:

1. W3C composite capability / preferences profile[13] (CC/PP);
2. WAP User Agent Profile (UAPROF);
3. Wireless Universal Resource File[14] (WURFL).

Il modulo determina l'identità di un particolare dispositivo mobile utilizzando il campo *header* della richiesta HTTP.

Tutti i dispositivi che sono conformi alla specifica UAPROF forniscono una descrizione CC/PP delle loro caratteristiche su un *repository server*. In questo modo server, gateway e proxy che ne fanno richiesta possono usare l'informazione per adattare il contenuto al dispositivo richiedente.

I profili descrivono le caratteristiche di un dispositivo con la sintassi *Resource Description Framework*[15] (RDF) che è in formato XML.

[13] http://www.w3.org/TR/CCPP-struct-vocab/
[14] http://wurfl.sourceforge.net/
[15] http://www.w3.org/RDF/

Quando il dispositivo mobile invia la richiesta al proxy server, invia contemporaneamente anche l'URL di dove si trova il suo profilo settando un campo dell'header della richiesta chiamato *X-Wap-Profile*.

Per esempio il campo *x_wap_profile* per un terminale mobile Nokia N9500 è:

x_wap_profile:"**http://nds1.nds.nokia.com/uaprof/N9500r100.xml**";

Il valore di *x_wap_profile* indica al proxy server dove poter trovare il profilo del dispositivo.

Il frammento di codice mostra una parte del profilo CC/PP per un terminale mobile:

```
<?xml version="1.0"?>
 <rdf:RDF
       xmlns:rdf= "http://www.w3.org/..."
       xmlns:prf="http://www.openmobilealliance.org/..."
       xmlns:mms="http://www.wapforum.org/..."
       xmlns:pss5="http://www.3gpp.org/...">
       <rdf:Description rdf:ID="Profile">
          ……
            <prf:component>
                 <rdf:Description rdf:ID="HardwarePlatform">
                 ……
                      <prf:PixelAspectRatio>1x1</prf:PixelAspectRatio>
                      <prf:PointingResolution>Pixel</prf:PointingResolution>
                      <prf:ScreenSize>640x200</prf:ScreenSize>
                      <prf:ScreenSizeChar>29x5</prf:ScreenSizeChar>
           <prf:StandardFontProportional>Yes</prf:StandardFontProportional>
                      <prf:SoundOutputCapable>Yes</prf:SoundOutputCapable>
                      <prf:TextInputCapable>Yes</prf:TextInputCapable>
                      <prf:Vendor>Nokia</prf:Vendor>
                      <prf:VoiceInputCapable>No</prf:VoiceInputCapable>
                   ……
                   </rdf:Description>
            </prf:component>
          ……
        </rdf:Description>
   </rdf:RDF>
```

Il modulo *Interface Manager* estrae l'informazione necessaria e se la memorizza, così che possa essere usata anche successivamente.

Esempio di codice per identificare la risoluzione del display:

```
<prf:ScreenSize>640x200</prf:ScreenSize>
```

5.2.2 Modulo *Page Manager*

Quando il modulo Interface Manager inoltra l'URL del sito web (es. la home page o qualsiasi altra pagina web) che deve essere trasformata, il modulo *Page Manager* recupera dal Server Web la pagina e la salva in una directory locale. Soltanto la pagina memorizzata in questa directory sarà sottoposta al processo di trasformazione.
Qui di seguito, verranno illustrate le singole parti che compongono il modulo.

Download Page

Il modulo *Download Page* salva la pagina web riferita all'URL in una directory (`CELL`).

Download Images

Quando il proxy riceve dal Web Server la pagina (versione per il desktop) e la memorizza nella directory, oltre alla pagina vengono salvate, in una directory locale al proxy chiamata *`images`*, anche tutte le immagini che essa contiene. In questo modo le pagine ottenute dalla trasformazione troveranno le immagini corrispondenti in quella directory.

5.2.3 Modulo *Reverse*

Il processo di trasformazione di una pagina web inizia con il *reverse engineering* della stessa. Il modulo *Reverse* sviluppato da [9] prende in input l'interfaccia utente di una singola pagina web, originariamente realizzata per un sistema desktop, e genera la corrispondente *descrizione logica* (CUI-desktop) che può essere successivamente trasformata per ottenere l'interfaccia utente per il dispositivo mobile destinatario.

Alla pagina web viene fatta corrispondere una presentazione (elemento `presentation`) e ogni tag XHTML viene mappato in un differente tipo di interattore concreto o combinazione di essi.

Dal momento che l'algoritmo di *reverse* lavora sull'albero DOM della pagina XHTML è necessario che la pagina sia ben formata (valida e conforme agli standard).

Spesso ciò non accade e quindi, prima di iniziare la fase di *reverse*, viene effettuato il parsing della pagina usando il parser Tidy[16] W3C che corregge i possibili *mismatch* e restituisce un DOM corretto.

L'algoritmo di reverse analizza ricorsivamente il DOM della pagina partendo dall'elemento *body* ed effettua una visita in profondità.

Per ogni tag che può essere direttamente mappato in un elemento concreto, una funzione specifica analizza il nodo corrispondente ed estrae l'informazione per generare l'interattore concreto o l'operatore di composizione appropriato (Tabella 5.1).

In relazione al DOM XHTML è possibile avere tre casi:

- Il nodo in questione è una foglia. Questo viene mappato in un interattore concreto ed inserito nella CUI-desktop. Ciò avviene, per esempio, per i tag <img>, <a> o <select> che causano la generazione dei seguenti interattori: *object*, *navigator* e *selection*.
- Il nodo corrisponde ad un operatore di composizione. In questo caso viene generato il codice relativo all'operatore di composizione ed effettuata una chiamata ricorsiva sul sottoalbero per generare gli elementi facenti parte di questo operatore di

[16] http://tidy.sourceforge.net/

composizione. Ciò avviene per il tag <form> che corrisponde all'operatore di composizione Relation.

- Il nodo non richiede la creazione di un'istanza di un interattore nell'interfaccia concreta. In tal caso nessun nuovo elemento viene aggiunto. Ciò può avvenire nel caso di paragrafi vuoti o
.

Corrispondenza tra elementi XHTML e gli elementi della CUI-desktop:

Elemento X/HTML	**Elemento CUI-desktop**
Unordered List	Grouping
Ordered List	Ordering
Div	Grouping
Fieldset	
Form	Relation
Input checkbox	Single select
Input radio	Multiple select
Select	Single / multiple select
Table	Description
Input text	Textfield
Input button	Navigator
Anchor	
Input reset	Activator
Input submit	
Text	Text
Image	Object(Image)
Text + Image	Description
Text + Anchor	InteractiveDescription

Tabella 5.1: Trasformazione da elemento XHTML a elemento della descrizione logica desktop

Nel caso in cui un file CSS è associato alla pagina analizzata, per ogni elemento che può essere caratterizzato da una definizione di stile (come per esempio colore dello sfondo, lo stile del testo, il font del testo), l'algoritmo determina le diverse proprietà di stile nel file CSS e usa questa informazione per effettuare una descrizione completa del corrispondente interattore.
Esempio di reverse engineering dell'elemento XHTML … : al tag XHTML viene fatto corrispondere un interattore <text> nella CUI-desktop (figura 5.3).

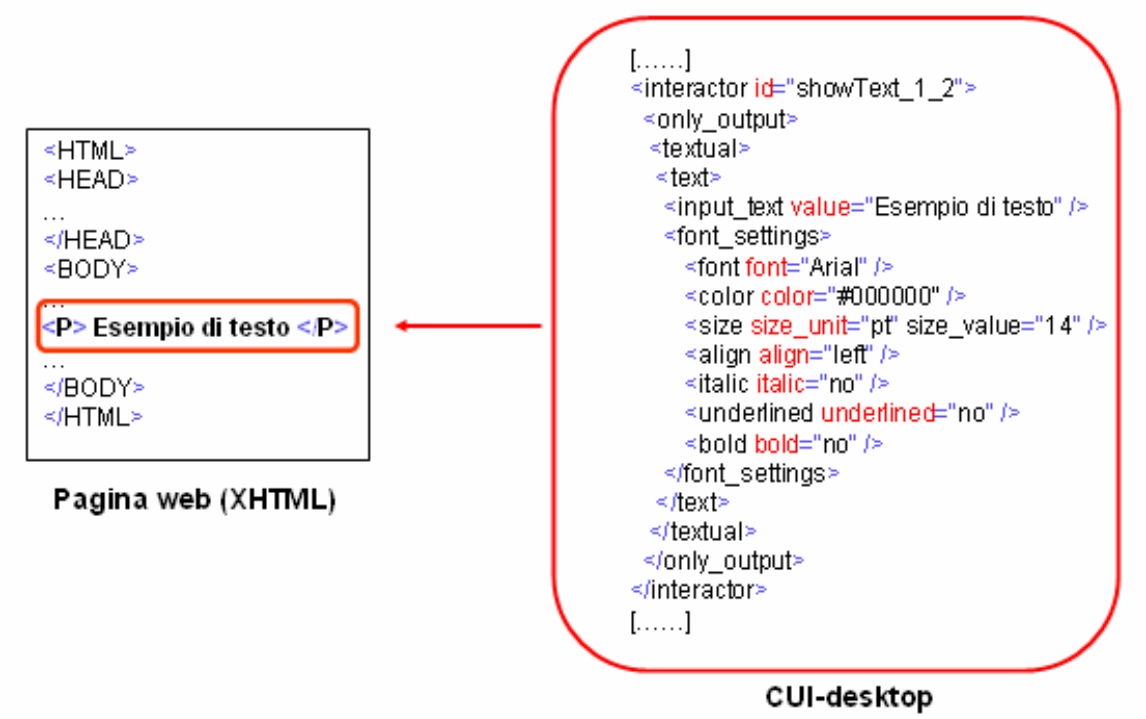


**Figura 5.3: Reverse engineering del tag .. **

5.2.4 Modulo *Redesign*

Il modulo *Redesign* trasforma la descrizione logica di un'applicazione interattiva desktop (ottenuta attraverso il modulo *Reverse*) nella descrizione logica per il dispositivo mobile prendendo in considerazione l'informazione semantica e i limiti delle risorse disponibili nel dispositivo [7].

Il modulo esegue l'algoritmo di *Cost-based Semantic Redesign*, discusso nel capitolo 4, prendendo in considerazione il costo in termini di spazio occupato dagli elementi che compongono l'interfaccia utente dell'applicazione desktop (testo, immagini, bottoni).

5.2.5 Modulo *Pages Generator*

Il modulo *Pages Generator* [10] è in grado di generare automaticamente l'interfaccia utente dell'applicazione interattiva per il dispositivo mobile destinatario a partire dall'Interfaccia Utente Concreta (CUI-mobile) creata dal modulo *Redesign*.
Il modulo in questione scandisce tutti gli elementi dell'albero DOM della CUI-mobile e da essi crea l'interfaccia utente.
Per ogni elemento `presentation` viene creata una pagina XHTML nella quale viene scritto il codice generato.
Le pagine generate contengono, in particolare, due diversi tipi di link: il primo permette di navigare tra le nuove pagine generate, il secondo esprime il collegamento ad ulteriori pagine dell'applicazione desktop che non sono state ancora visitate.
Per poter effettuare la navigazione da terminale mobile è necessario che il modulo *Pages Generator* modifichi opportunamente i link.

Ecco la struttura dei link nei due possibili casi:

- **Tipo 1**: link verso pagine splittate.
  Esempio: …

- **Tipo 2**: ridirezione automatica verso il proxy-server.
  Esempio:
  …
  è sostituito con
  ...

In quest'ultimo caso il link è costruito in modo da ripassare dal proxy server con la nuova pagina.

5.3 Processo di trasformazione delle pagine

Gli utenti interagiscono con il sistema attraverso un qualsiasi browser web installato sui loro terminali mobili e non hanno alcun bisogno di configurare il dispositivo prima di utilizzare l'applicazione.

L'utente in particolare deve far partire il browser del dispositivo e digitare l'URL del proxy server che ospita l'applicazione.

Ciò che gli verrà restituito è la pagina iniziale dell'applicazione (figura 5.4).

Figura 5.4: Pagina iniziale del prototipo di sistema

Dopo che l'utente ha avuto accesso alla pagina iniziale dell'applicazione, inserisce l'indirizzo della pagina web che vuole visitare e preme il bottone di "Navigate".

Il proxy server recupera la pagina, la "splitta" e restituisce la prima delle varie pagine ottenute.

Durante il processo di trasformazione (figura 5.5), una pagina web passa attraverso vari stadi:

a) L'utente effettua una richiesta di una pagina web da terminale mobile che arriva al proxy che a sua volta la inoltra al server web.

b) Il proxy salva la pagina ricevuta dal server in una directory locale (`CELL`) e le eventuali immagini contenute in essa in un'altra directory (`images`).

c) La pagina XHTML passa attraverso il modulo *Reverse* che effettua il *reverse engineering* della stessa. In questo modo si ottiene la descrizione logica della pagina web (CUI-desktop) conforme alla DTD-desktop, memorizzata in un file XML.

d) Il file CUI-desktop generato viene preso come input dal modulo *Redesign* che applica l'algoritmo di *Cost-based Sematic Redesign* e genera la nuova CUI-mobile per il dispositivo mobile destinatario.

e) La CUI-mobile ottenuta passa al modulo *Pages Generator* che genera le pagine XHTML che vengono salvate in una directory (`FUI`). Subito dopo la prima delle pagine ottenute viene inviata al terminale mobile per essere visualizzata dall'utente.

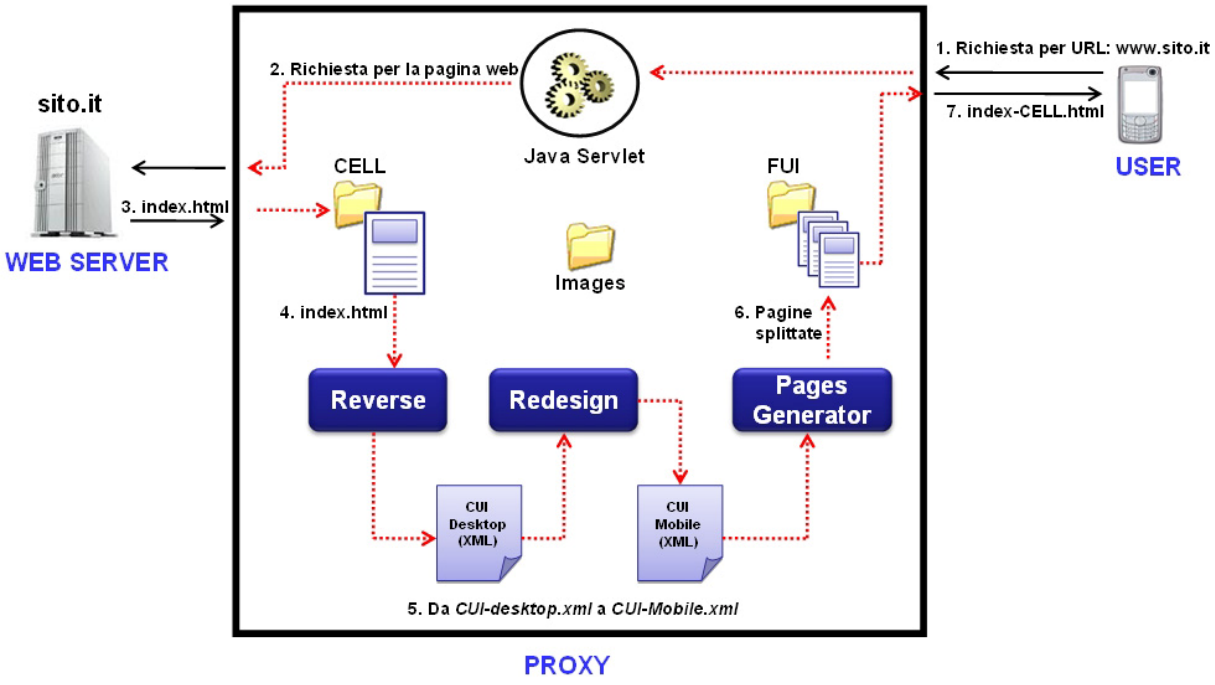


Figura 5.5: Processo di trasformazione delle pagine web

5.4 Esempi illustrativi

In questo paragrafo vengono presentati due esempi di come il prototipo di sistema funziona con due pagine web desktop.

Esempio #1

Consideriamo ora un esempio di esecuzione del sistema.
L'esempio mette in evidenza il lavoro che viene eseguito dal processo di *Redesign* nel contesto della realizzazione delle presentazioni per terminali mobili a partire dal una pagina web originariamente progettata per una piattaforma desktop.
Supponiamo che l'utente acceda alla pagina usando un terminale mobile. Il proxy server intercetta la richiesta, recupera la pagina web dal Server Web e applica il processo di *reverse-redesign* per generare una versione della pagina adatta alle risorse del terminale mobile. Prima del *reverse engineering*, il proxy server filtra (se presenti) alcuni elementi della pagina (es. tag *applet*, *embed*, ecc) e li rimuove.
La figura 5.6 mostra la pagina web desktop come vista da un browser con gli operatori di composizione associati.

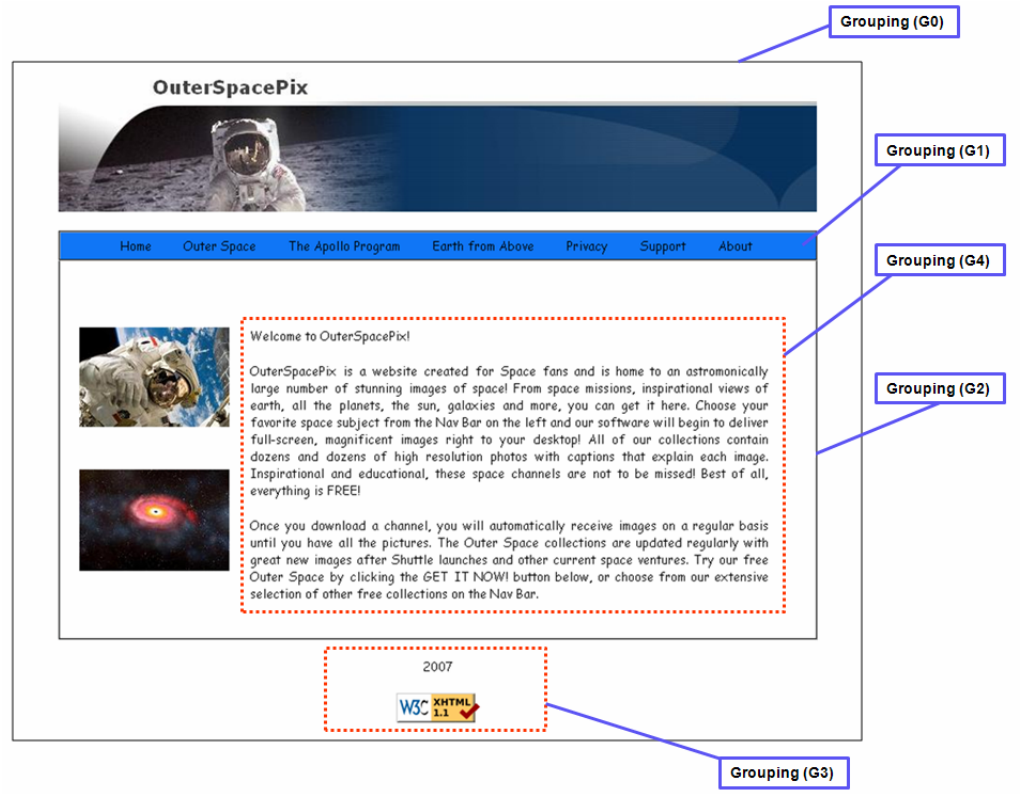


Figura 5.6: Pagina web desktop con identificazione dei relativi operatori di composizione

Il processo di *Redesign* inizia con la specifica XML della CUI-desktop creata dal modulo *Reverse*. La descrizione della CUI-desktop contiene interattori (come text, image) e operatori di composizione (*grouping*) che definiscono come strutturare gli interattori.

La figura 5.7 mostra la struttura dell'albero DOM della CUI-desktop corrispondente alla pagina web.

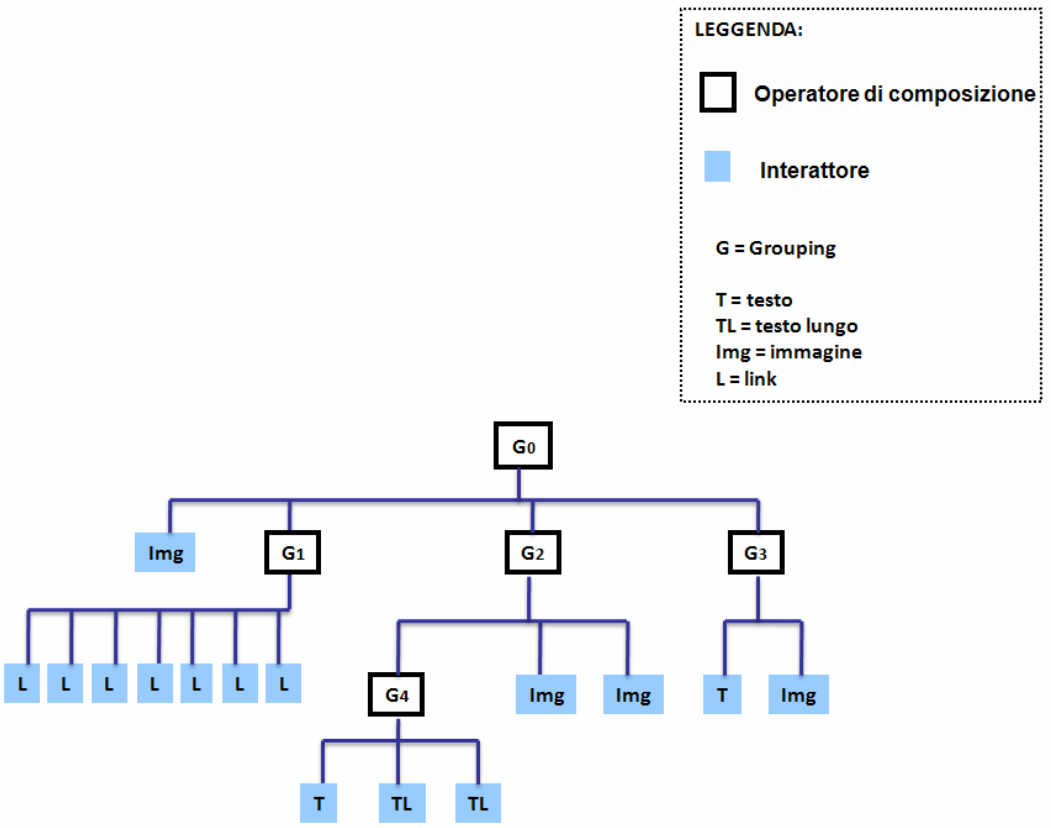


Figura 5.7: Struttura dell'albero DOM della CUI-desktop

Nella figura 5.7 vediamo che c'è un operatore *grouping* (G0) che coinvolge tutti gli elementi della pagina: un interattore immagine (Img) e tre operatori di grouping (G1, G2, G3).

In generale il *grouping* serve ad identificare un raggruppamento di elementi che condividono particolari caratteristiche strutturali.

La presentazione desktop contiene 15 interattori che richiedono un'occupazione di risorse troppo grande per essere contenuta in una singola presentazione per terminale mobile.

Il modulo *Redesign* eseguendo l'algoritmo di *Cost-based Semantic Redesign* divide la presentazione desktop dell'esempio in cinque presentazioni per il terminale mobile tenendo conto del costo in termini di spazio occupato dagli elementi dell'interfaccia web.

Considerando la struttura dell'albero DOM della CUI-desktop, l'algoritmo esegue una visita in profondità (DFS) partendo dalla sua radice e genera le presentazioni per *mobile* inserendo

gli elementi presenti della pagina originaria tenendo conto delle regole di trasformazione e del costo che la presentazione per mobile può supportare.

Per prima cosa viene calcolato il costo di ogni interattore e operatore di composizione.

La procedura `CalculateCost()` esegue una visita in profondità della struttura ad albero partendo dalla radice.

Il costo di un operatore di composizione è dato dalla somma dei costi dei nodi figli più, se presente, il costo della "cornice", mentre quello di un interattore è calcolato direttamente tenendo conto delle regole di trasformazione (fase *TransformCUI* dell'algoritmo) sulle quali si basa il sistema, cioè, in caso di trasformazione, il costo è quello dell'interattore trasformato.

Alla fine di questa prima fase, ogni interattore e operatore di composizione avrà assegnato il proprio costo (figura 5.8).

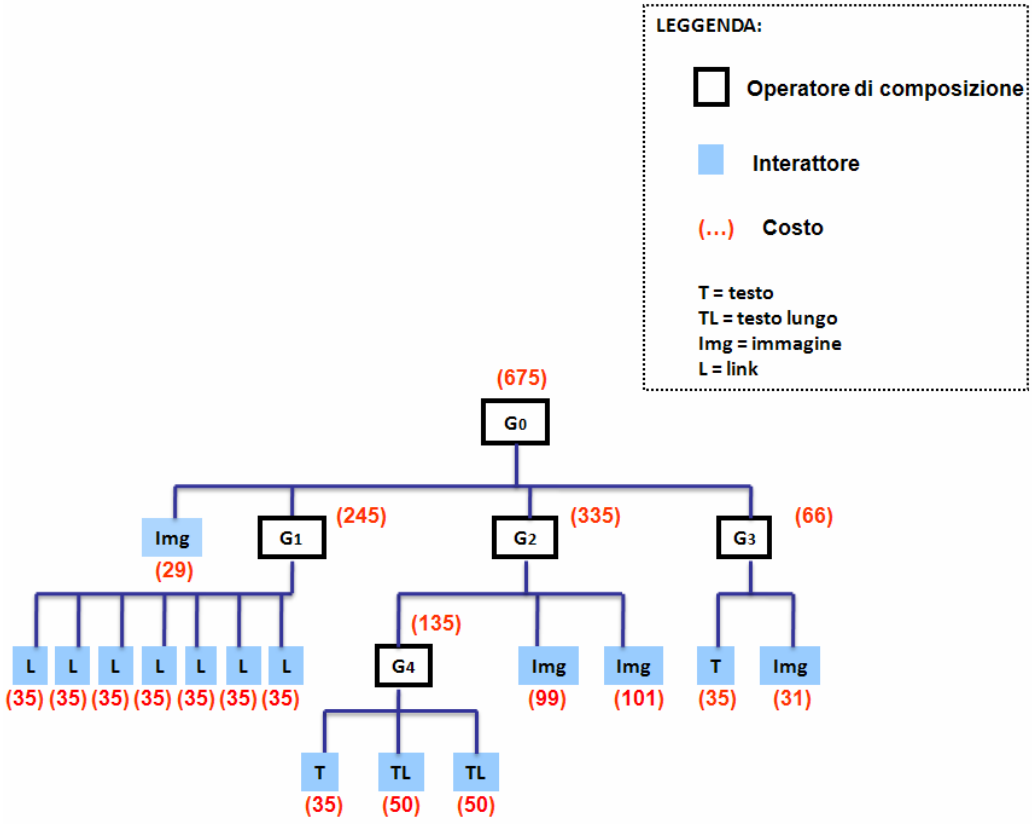


Figura 5.8: Interattori e operatori di composizione della CUI-desktop con relativi costi

A questo punto viene eseguito l'algoritmo di *page splitting* (fase *SplittingCUI*) che decide come gli interattori devono essere disposti nelle presentazioni per *mobile* tenendo conto sia dei costi degli interattori e operatori di composizione sia delle regole di splitting.
Per prima cosa l'algoritmo determina se ci sono degli operatori di composizione che devono essere ridotti a link.
Di seguito viene mostrato l'applicazione del metodo `reduceAtLink()` al `ComplexElem` G_0 (figura 5.9):

1. Calcola il costo di G_0 (es. 29px + 245px + 335px + 66 px = 675px)
2. Poichè il costo di G_0 > `COSTO_MAX_PAGE_MOBILE` determina il `ComplexElem` figlio di G_0 di costo massimo (cioè G_2)
3. Aggiorna il suo costo: costo(G_2) = 35px
4. Setta un booleano per indicare che G_2 deve essere ridotto a link
5. Ricalcola il costo di G_0 (es. 29px + 245px + 35px + 66 px = 375px)
6. Poichè il costo di G_0 > `COSTO_MAX_PAGE_MOBILE` determina il `ComplexElem` figlio di G_0 di costo massimo (cioè G_1)
7. Aggiorna il suo costo: costo(G_1) = 35px
8. Setta un booleano per indicare che G_1 deve essere ridotto a link
9. Ricalcola il costo di G_0 (es. 29px + 35px + 35px + 66 px = 165px)
10. Crea la prima presentazione
11. Inserisce i figli di G_0 nella presentazione:
    12. Quando incontra l'interattore immagine (Img) lo inserisce direttamente
    13. Quando incontra G_1 e G_2: sostituisce i `ComplexElem` con un link e chiama ricorsivamente la funzione su quei nodi.
    14. Quando incontra G_3: inserisce il `ComplexElem` nella presentazione corrente e chiama ricorsivamente la funzione su quel nodo.

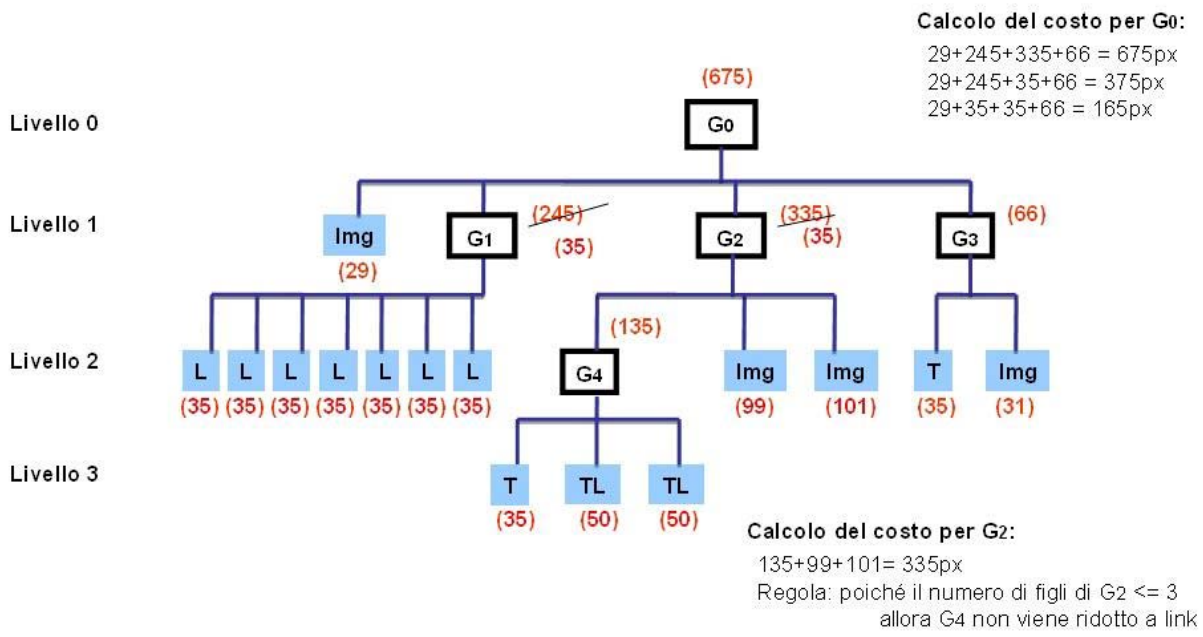


Figura 5.9: Esecuzione del metodo `reduceAtLink()`

La figura 5.10 mostra il risultato della trasformazione.

La pagina web desktop originaria è stata "splittata" in cinque presentazioni/pagine per il dispositivo mobile.

Analizzando la *Pagina 1* si può notare che contiene due link che corrispondono agli operatori di composizione (`ComplexElem`: G1 e G2) che sono stati ridotti a link, mentre il `ComplexElem` G3 è stato inserito per intero.

Nella *Pagina 3* possiamo notare la trasformazione dell'immagine: è stata ridimensionata per risparmiare dello spazio.

Per quanto riguarda il testo, è stato spezzato in più parti sostituendolo nella *Pagina 3* con una linea di testo e un link al restante.

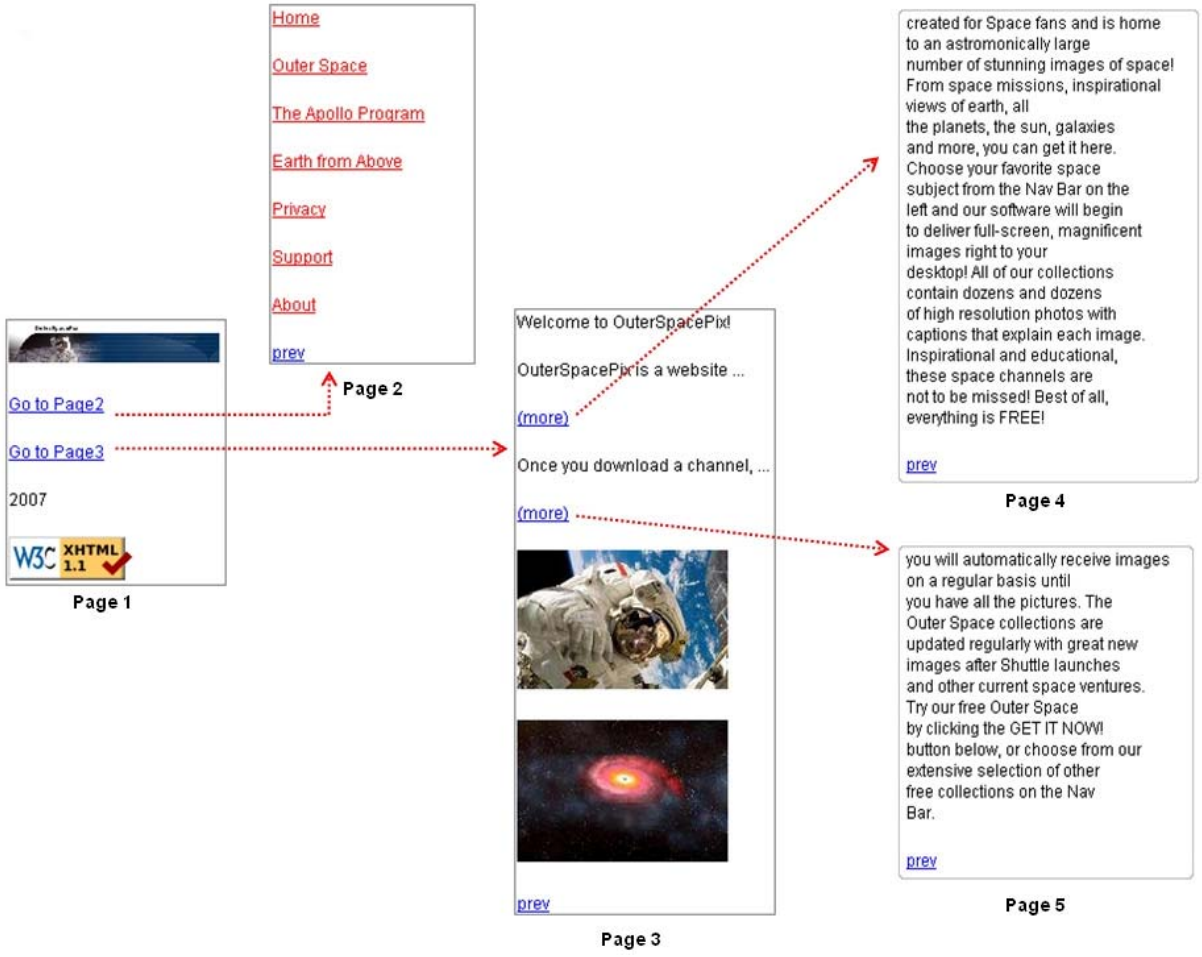


Figura 5.10: Pagine per mobile che risultano dalla trasformazione della pagina desktop

Esempio #2

Consideriamo ora un altro esempio per vedere cosa accade in presenza di un testo lungo e di una form.

La figura 5.11 mostra l'interfaccia utente dell'applicazione desktop.

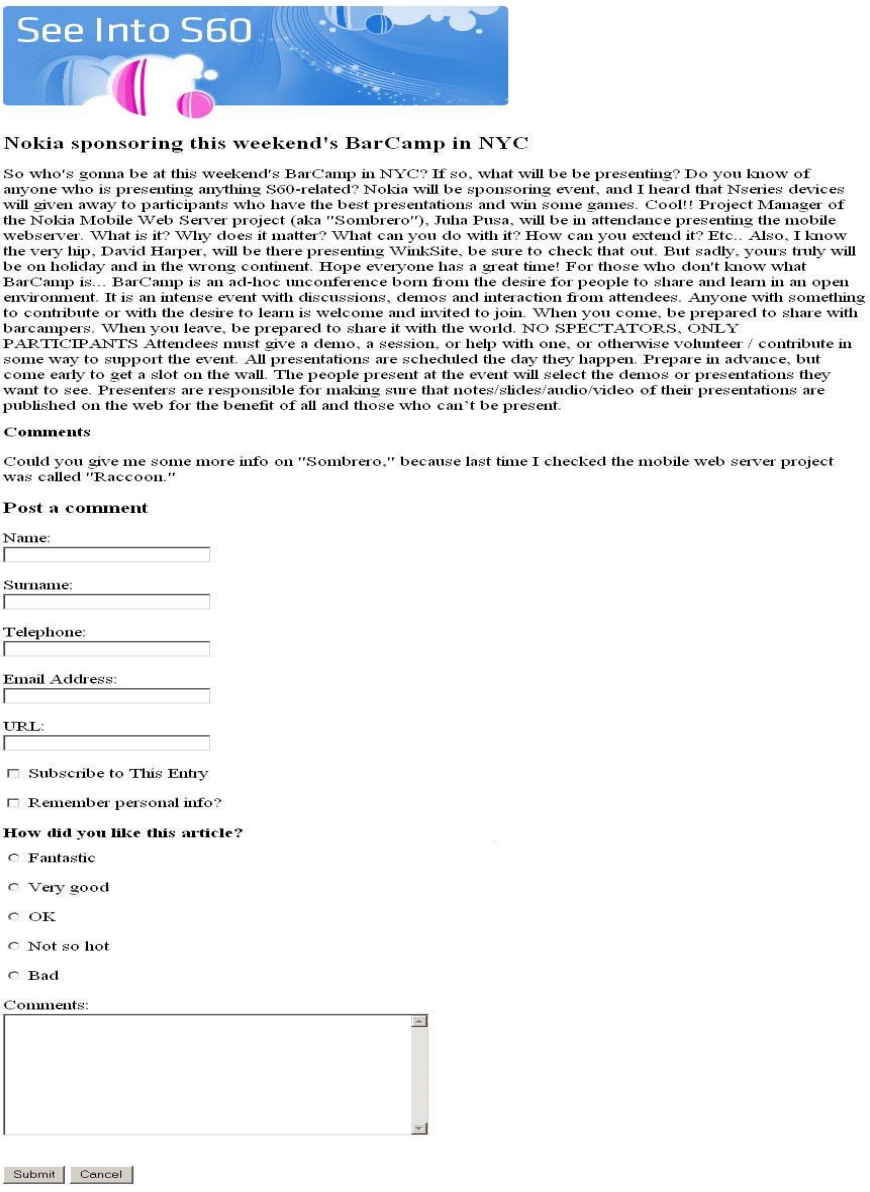


Figura 5.11: Interfaccia utente dell'applicazione interattiva desktop

La figura 5.12 mostra il risultato della trasformazione che ha portato alla creazione di sette pagine per il dispositivo mobile.

Il testo lungo è stato spezzato su più pagine con la particolarità che nella prima pagina è stata inserita la prima riga di testo e un link (*more*) al restante testo con la possibilità di navigarlo attraverso i link (*next* e *prev*).

La form è stata spezzata su più pagine (*Pagine 5, 6, 7*) e i due bottoni *Submit* e *Cancel* sono stati inseriti nella prima pagina.

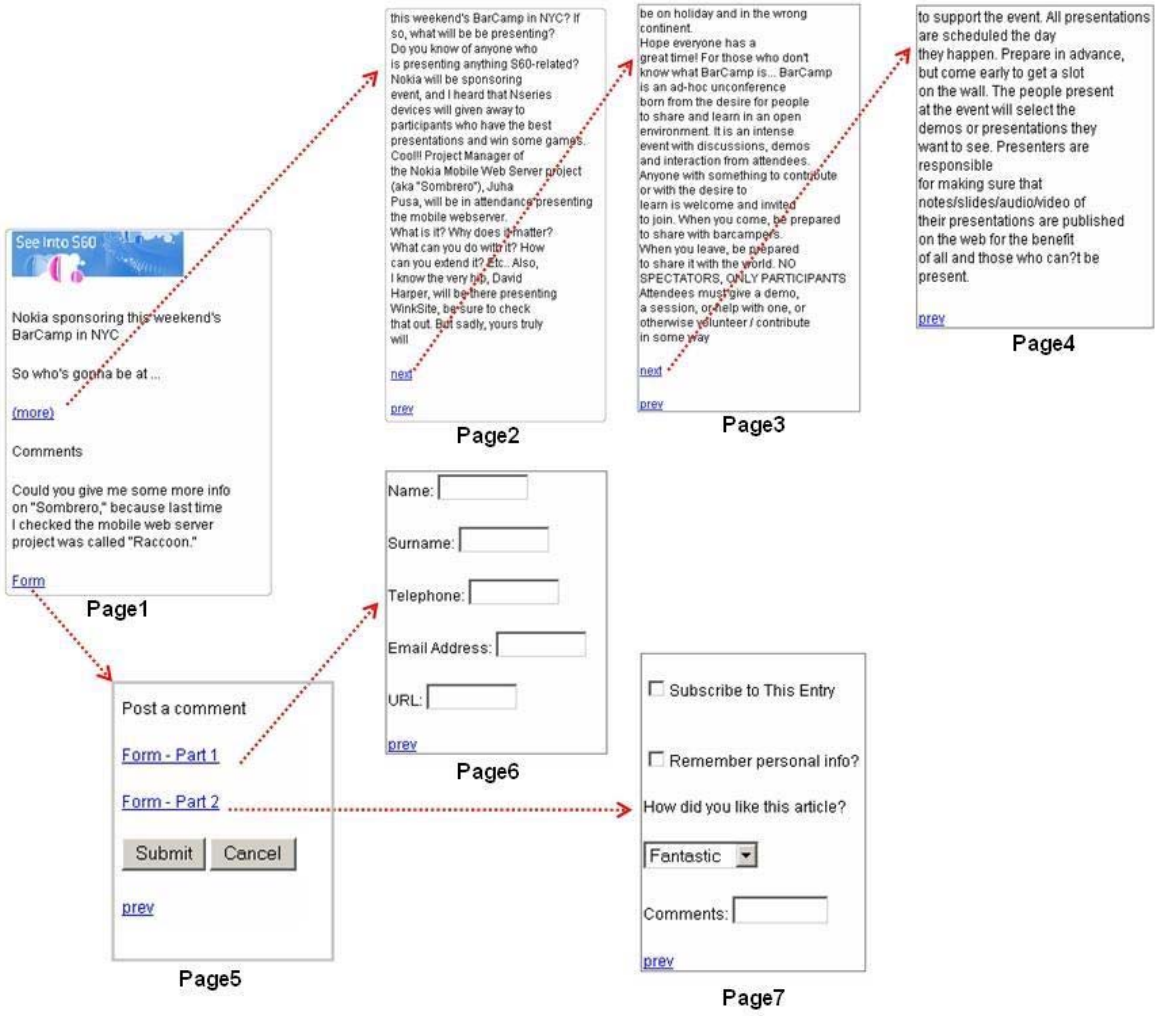


Figura 5.12: Pagine per mobile che risultano dalla trasformazione della pagina desktop

5.5 Valutazioni

Dopo la realizzazione del prototipo di sistema è stato eseguito un test per valutare l'usabilità delle pagine web ottenute dal processo di trasformazione.

Il test ha visto il coinvolgimento di un gruppo eterogeneo di volontari. L'età degli utenti era compresa tra 23 e 41 anni, con una media di 28,1 ed erano per l'80% laureati. Il 50% aveva precedentemente usato un terminale mobile per navigare su Internet. Gli utenti inoltre avevano una buona esperienza di interfacce utente per PC, ma nessuna conoscenza di tool di redesign.

Durante il test agli utenti sono state mostrate diverse pagine web originariamente implementate per la piattaforma desktop e successivamente, dopo averle analizzate, hanno utilizzato tre tool di redesign: Google[17], SemanticTransformer (il prototipo di sistema), Skweezer[18] per ottenere una versione della stessa pagina riprogettata per la piattaforma mobile.

La visualizzazione della pagina prodotta per la piattaforma mobile è stata simulata tramite un opportuno ridimensionamento della finestra del browser.

L'utente ha dovuto confrontare i risultati prodotti dai vari tool di redesign proposti e valutare vantaggi e svantaggi dei sistemi considerati.

All'inizio del test gli utenti hanno letto un'introduzione su natura e scopi del sistema.

La prima parte del test ha riguardato la valutazione della qualità dei risultati ottenuti dalla trasformazione delle pagine desktop eseguita dai tre tool: agli utenti è stato in particolare chiesto di navigare/analizzare le pagine prodotte.

I tool con cui si è confrontato il prototipo realizzato presentavano delle caratteristiche comuni. Le differenze erano poche (ad esempio un diverso uso dei CSS, a volte una diversa formattazione delle pagine) e dipendevano dalle pagine analizzate.

La figura 5.13 mostra una delle pagine web usate per il test.

[17] http://www.google.com/xhtml
[18] http://www.skweezer.net

Figura 5.13: Interfaccia utente per piattaforma desktop

Successivamente gli utenti hanno visionato ed interagito con le pagine ottenute dalla trasformazione dai tre tool (figura 5.14).

Dalla figura 5.14 si può osservare che le pagine prodotte dai tool di Google e Skweezer presentavano delle caratteristiche comuni, richiedendo soltanto lo scrolling verticale per la navigazione della pagina (*Page 1*), mentre il SemanticTransformer creava più pagine (*Page 1*, *Page 2*, *Page 3*) e riduceva lo scrolling verticale.

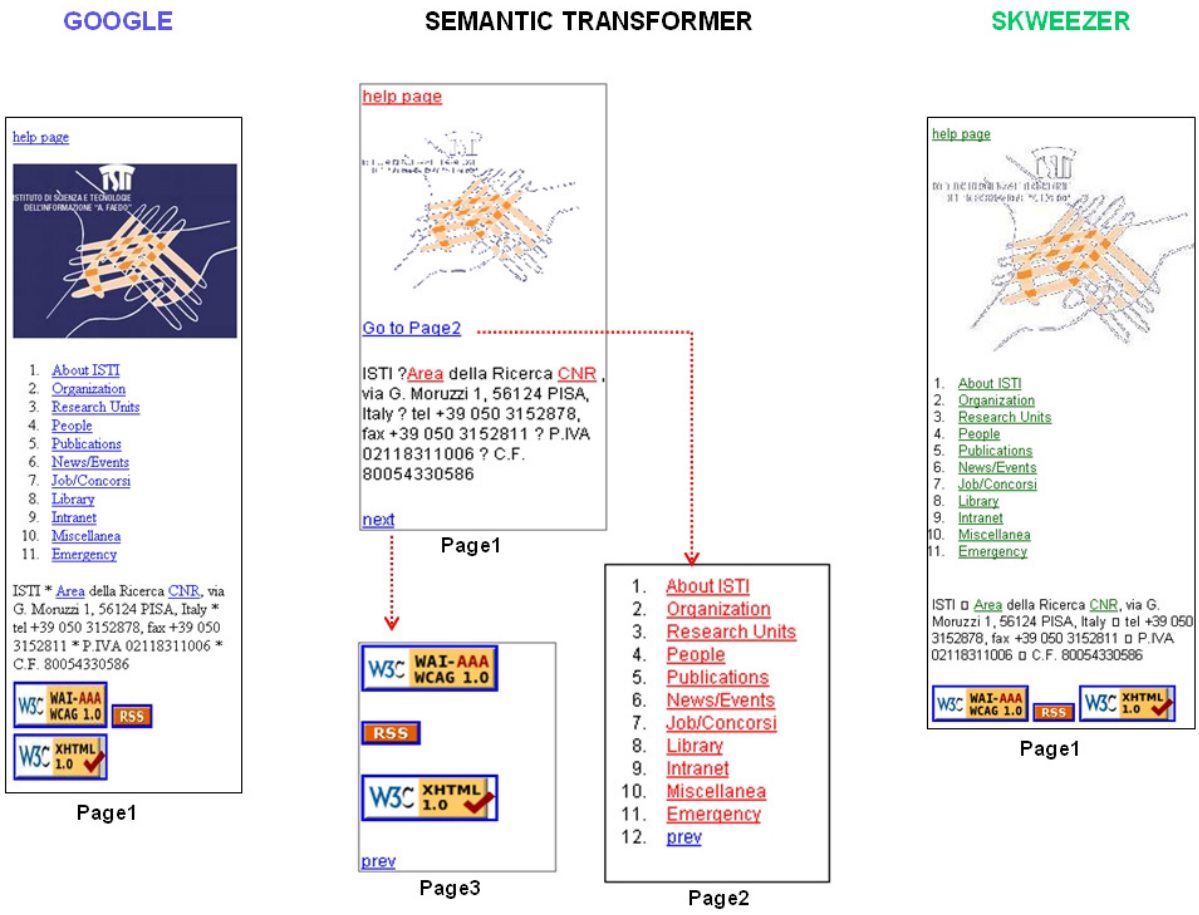


Figura 5.14: Pagine per mobile ottenute dai tre tool

Dopo aver testato tutte le pagine web, ad ogni utente è stato chiesto di compilare un questionario di valutazione.

Le domande riguardavano:

1. La facilità di mettere in corrispondenza la pagina per la piattaforma mobile con la relativa pagina per la piattaforma desktop.
2. Il modo in cui i tool di redesign supportano la trasformazione delle *immagini* presenti nella pagina desktop.
3. Il modo in cui i tool di redesign supportano la trasformazione dei *testi lunghi* presenti nella pagina desktop.

4. La trasformazione di altri oggetti dell'interfaccia (ad esempio: *radio button*, *text area*, *drop down list*).
5. L'usabilità complessiva delle pagine prodotte dai vari tool analizzati durante il test.
6. Esprimere un giudizio riguardo alla divisione (o *splitting*) di una pagina desktop in più pagine sulla piattaforma mobile (soltanto per il SemanticTransformer).
7. La convenienza dello *splitting* di una pagina desktop in più pagine per la piattaforma mobile.

Alla maggior parte delle domande l'utente doveva rispondere attribuendo un valore variabile da 1, per indicare il punteggio più basso (difficile/non efficace), a 5, per dare un parere positivo (facile/efficace), ed esprimere un eventuale commento.
I risultati sono mostrati nella figura 5.15.

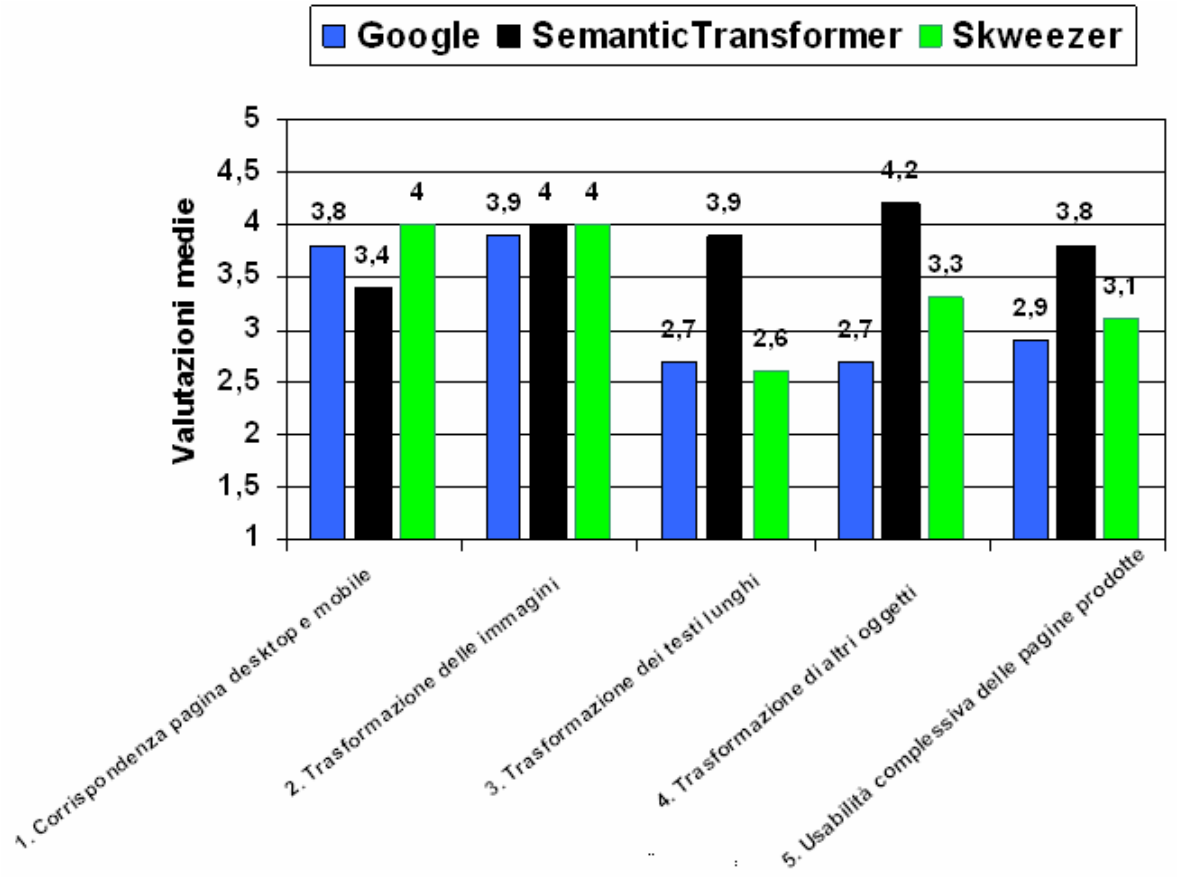


Figura 5.15: Media dei punteggi che gli utenti hanno assegnato

Dando uno sguardo alle risposte date alla domanda 1, come si può vedere dal grafico, le pagine prodotte da Google e Skweezer sono risultate di più facile comprensione (corrispondenza lineare tra la versione desktop e mobile), mentre quelle del SemanticTransformer potevano disorientare l'utente, soprattutto in presenza di molti link.

Il *page splitting*, utilizzato dal SemanticTransformer, ha reso più difficile mettere in corrispondenza le pagine delle due piattaforme rispetto alla tecnica dello scrolling verticale, in quanto gli utenti dovevano abituarsi alla suddivisione delle pagine.

Per quanto riguarda la trasformazione delle immagini (domanda 2) il giudizio espresso è stato quasi uguale per tutti e tre i tool. Confrontando le immagini prodotte per l'esempio considerato in figura 5.14 sembra che le immagini prodotte perdano il colore blu in due dei tre tool. In realtà, non è così in quanto l'immagine originaria è come mostrata in figura 5.16 ed è Google che vi aggiunge come sfondo il blu.

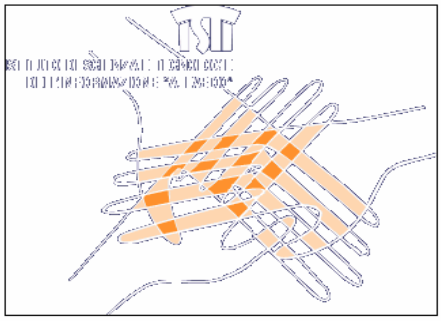


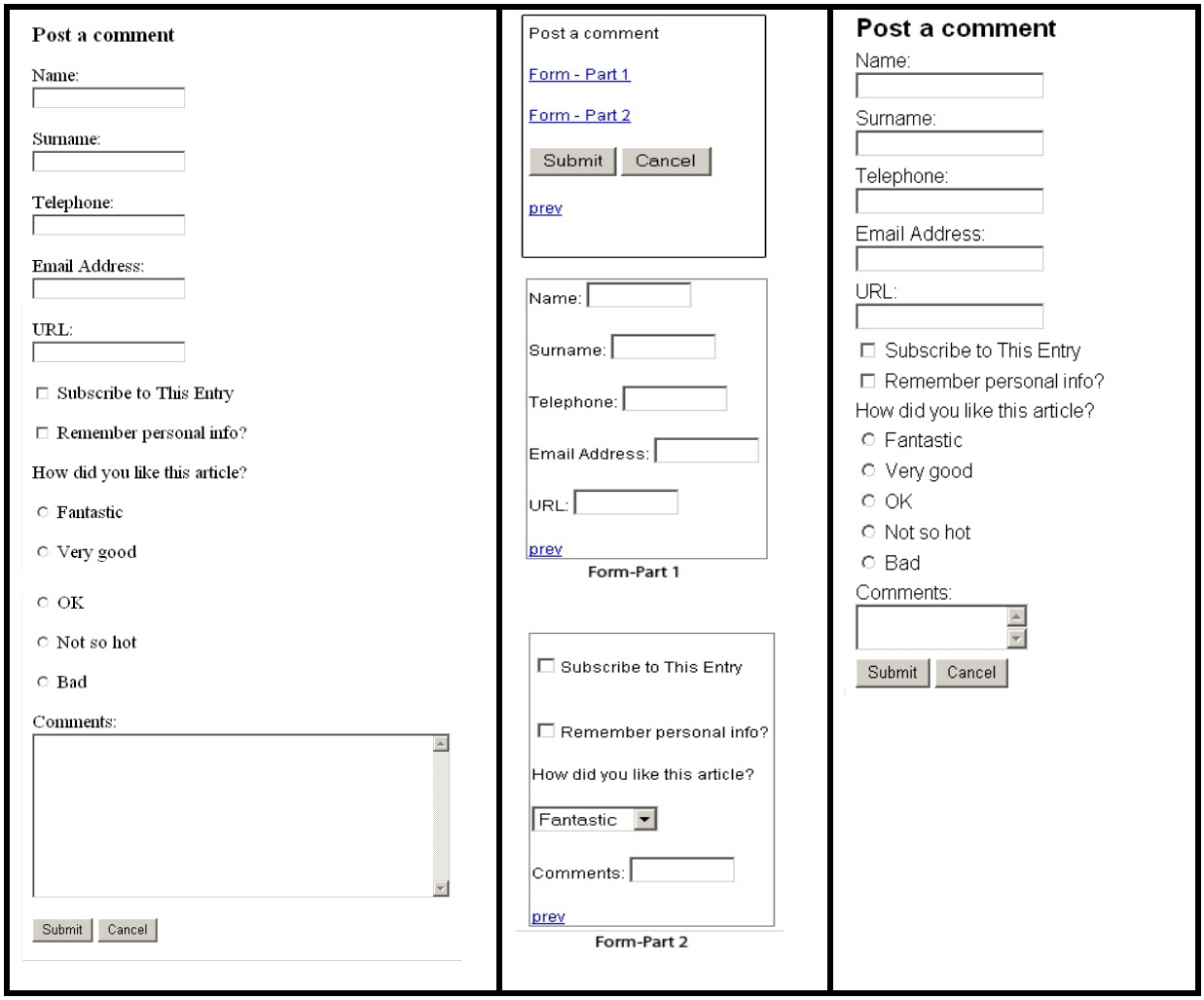
Figura 5.16: Immagine originaria

Alla domanda riguardante la trasformazione dei *testi lunghi* (domanda 3) la maggior parte degli utenti ha risposto esprimendo una preferenza per la suddivisione del testo in più parti rispetto alla presentazione compressa e in verticale degli altri due tool. Il suggerimento dato dagli utenti è stato quello di aumentare il numero di parole prima di inserire il link *"more"* (ad esempio 4 o 5 righe da poter leggere per avere un'idea del contenuto).

Per quanto riguarda la trasformazione degli altri oggetti dell'interfaccia (domanda 4), l'utente ha gradito la trasformazione di alcuni elementi della form (ad esempio il *radio button* che veniva trasformato in un *drop down list* o la *textarea* in un *textfield*).

Dall'analisi dei commenti espressi è emersa la preferenza per una visione compatta della form (tutta in una pagina) e che tra Google e Skweezer, quest'ultimo è stato gradito di più perché

presentava gli elementi della form in maniera più compatta, grazie ad un utilizzo efficace dei CSS (figura 5.17).



(a) Google **(b) Semantic transformer** **(c) Skweezer**

Figura 5.17: Form visualizzata dai tre tool

Complessivamente (domanda 5) è stato gradito l'approccio del *page splitting* rispetto a quello dello scrolling verticale. I commenti hanno riguardato in particolare l'aumento del numero di richieste che il terminale mobile deve fare al proxy a causa dei molti *split* (bisognerebbe capire se è più conveniente avere molte richieste di pagine che occupano pochi *Kb* oppure un'unica richiesta e scaricare soltanto una volta la pagina).

Alla domanda 6, che chiedeva di dare un giudizio sulla divisione (o *splitting*) di una pagina desktop in più pagine sulla piattaforma mobile (per il SemanticTransformer), si è ottenuto un valore di 3,8. I suggerimenti a riguardo sono stati: inserire una "barra di percorso" all'inizio e alla fine della pagina per tener traccia della navigazione quando le pagine "splittate" sono molte; inserire in ogni pagina un link per tornare alla pagina principale e quando possibile dare l'opportunità di passare tra una tecnica e l'altra (cioè *page splitting* e scrolling verticale).
Gli utenti hanno soprattutto fatto notare che bisognerebbe dare nomi significativi ai link per spostarsi da una pagina all'altra (ad esempio al posto di *Go to page2* un nome significativo che dia un'idea abbastanza chiara del contenuto della pagina a cui punta).
Alla domanda 7, sulla convenienza dello *splitting*, il 70% degli utenti ha dato una risposta positiva.
Dopo il test e l'analisi dei risultati si è deciso di aumentare in presenza di *testi lunghi* il numero di parole prima di inserire il link *"more"* (figura 5.18).

Figura 5.18: Trasformazione del testo lungo prima e dopo il test di usabilità

Capitolo 6

Conclusioni e sviluppi futuri

La crescente disponibilità di dispositivi mobili ha cambiato il modo di accedere a Internet, non solo attraverso un personal computer fisso, ma ovunque con un dispositivo mobile.
La maggior parte delle applicazioni web potrebbe ora essere acceduta utilizzando molteplici piattaforme (ad esempio: telefono cellulare, pda, ecc.), diverse da quella del classico desktop PC.
Nonostante ciò, l'accesso a Internet tramite di esse, risulta tuttora limitato dal momento che la maggior parte dei contenuti web sono realizzati per computer desktop.
La presenza di diversi tipi di dispositivi pone una serie di problemi ai progettisti e agli sviluppatori di applicazioni web, dal momento che è molto costoso realizzare e mantenere aggiornate più versioni dello stesso sito web per ogni tipo di dispositivo.
A questo scopo, sta crescendo l'interesse per tool che consentono di riprogettare (*redesign*) automaticamente pagine scritte per una certa piattaforma originaria (*desktop*), allo scopo di accederle da una diversa piattaforma destinazione (*mobile*).
Nella tesi è stato presentato un approccio per l'adattamento automatico di applicazioni interattive desktop per dispositivi mobili. In particolare, è stato realizzato un prototipo di sistema che esegue il processo di trasformazione delle pagine web basandosi su delle regole.
La nuova versione dell'algoritmo di trasformazione, *Cost-based Semantic Redesign*, prende in considerazione gli aspetti semantici (derivati dalla descrizione logica dell'interfaccia utente desktop) e il costo in termini di spazio occupato sulla pagina dagli elementi che compongono l'interfaccia, per garantire agli utenti una maggiore soddisfazione dal punto di vista dell'usabilità delle pagine.
Rispetto ai tool esistenti, che presentano la pagina web desktop ai dispositivi mobili in una forma lunga e stretta, l'utilizzo dell'informazione semantica è di utilità per individuare il modo migliore per dividere la pagina. Inoltre è stata proposta una tecnica per accedere

facilmente contenuti web con controlli tipo bottoni/link al fine di eliminare lo scrolling orizzontale e di ridurre quello verticale (dai risultati ottenuti dal test utente è emerso il gradimento dell'approccio del *page splitting*).
Il prototipo di sistema realizzato tuttavia presenta alcuni limiti:

- la complessità della struttura delle pagine web è un fattore critico sia per il processo di *reverse engineering* (risulta difficile trasformare la pagina originaria nella corrispondente descrizione logica), sia per il processo di *redesign* (si possono ottenere pagine con molti link).

- Non gestisce ancora tutti i tipi di contenuto web. Per esempio, le pagine contenenti animazioni, effetti grafici pubblicitari (banner) e frame non vengono gestiti: elementi come `<applet>`, `<embed>`, `<javascript>` sono filtrati all'inizio del processo e non sono quindi presenti nelle pagine generate.

- Per il momento l'applicazione usa il protocollo UAPROF per ricavare l'informazione della risoluzione del display senza utilizzarla, ma in futuro si potrà rendere le regole di trasformazione parametriche rispetto a tali valori.

Il test di usabilità del sistema ha permesso di ricavare indicazioni e suggerimenti utili per migliorare il prototipo di sistema.
Sviluppi futuri del prototipo potrebbero riguardare il perfezionamento del processo di *reverse engineering* per riuscire a gestire pagine web complesse per trasformarle nelle corrispondenti descrizioni logiche (ad esempio pagine con codice javascript e annidamento profondo dei tag XHTML).
Si potrebbe poi cercare di fare un'analisi del contenuto o, quando possibile, di estrapolare dai tag dell'XHTML dei nomi significativi da potergli dare ai link che collegano le pagine.
Altri sviluppi potrebbero, inoltre, riguardare il miglioramento dell'aspetto delle pagine generate attraverso un uso efficace dei CSS ed il mantenimento nelle pagine prodotte, nei limiti del possibile, dello stile della pagina web desktop originaria, per garantire una maggiore corrispondenza tra le due versioni.

Bibliografia

[1] G. Mori, F. Paternò. Automatic Semantic Platform-dependent Redesign. Smart Objects and Ambient Intelligence, pp.177-182, Grenoble, 2005.

[2] S. Berti, F. Correani, F. Paternò, C. Santoro. The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels. Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, pp.103-110, 2004.

[3] R. Bandelloni, G. Mori, F. Paternò. Dynamic Generation of Web Migratory Interfaces. Proceedings Mobile HCI'2005, Salzburg, ACM Press, pp. 83-90, 2005.

[4] F. Correani, G. Mori, F. Paternò. Supporting Flexible Development of Multi-Device Interfaces, Proceedings EHCIDSVIS'04, Hamburg, July 2004, Springer Verlag, Lecture Notes Computer Science.

[5] R. Bandelloni, G. Mori, F. Paternò. Automatic User Interface Generation and Migration in Multi-Device Web Environments. *Technical Report*, 2006.

[6] F. Paternò, G. Mori. Interaction Semantics Consistency in Cross-Platform Design. Proceedings of CHI 2006 Workshop.

[7] G. Mori, F. Paternò, C. Santoro, A. Scorcia. Migrating Web Applications through Declarative Models. 2007.

[8] R. Bandelloni, G. Mori, F. Paternò, Z. Salvador. A Service-Oriented Environment for Supporting Multimodal Web User Interface Migration, 2005.

[9] R. Bandelloni, F. Paternò, C. Santoro. Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments. Proceedings EIS'07, Salamanca, LNCS Springer Verlag, March 2007.

[10] G. Mori, F. Paternò, C. Santoro. Design and Development of Multi-Device User Interfaces through Multiple Logical Descriptions. IEEE Transactions on Software Engineering, Vol.30, N.8, pp.507-520, IEEE Press, 2004.

[11] Youn-Sik Hong e Ki-Young Lee. A Real-Time Web Contents Adaptation for Mobile User. M. Gavrilova et al. (Eds.): ICCSA 2006, LNCS 3981, pp. 249 – 258, 2006.

[12] Bickmore T., et al.. Web PageFiltering and Re-Authoring for Mobile Users. Computer Journal special issue on Mobile Computing, vol. 42, no. 6, 1999, pp. 534-546.

[13] Patrick Baudisch, Xing Xie, Chong Wang, Wei-Ying Ma. Collapse-to-zoom: Viewing web pages on small screen devices by interactively removing irrelevant content. ACM Symposium on User Interface Software and Technology (UIST '04), Santa Fe, USA, October 2004.

[14] Trevor J., Hilbert D., Schilit B., Koh T.. From Desktop to Phonetop: A UI For Web Interaction On Very Small Devices. ACM symposium on User interface software and technology, p. 121-130, 2001.

[15] D. H. Xiangye Xiao, Qiong Luo, H. Fu. Slicing*-tree based web page transformation for small displays. CIKM, 2003.

[16] Antero Kivi. MOBILE BROWSING, *Technical Report*, 2007. http://www.tml.tkk.fi/Opinnot/T-109.7510/2007/reports/MobileBrowsers.pdf

[17] Evgeniya Georgieva, Tsvetan Hristov. Design of an e-Learning Content Visualization Module. 3rd E-Learning Conference Coimbra, Portugal, 2006.

[18] T. Laakko, T. Hiltunen. Adapting Web Content to Mobile User Agents. IEEE Internet Computing, Vol. 9, Issue 2, pp 46-53, March-April 2005.

[19] Web Content Adaptation. White Paper. 2004. http://www.medialab.sonera.fi/workspace/WebContentAdaptationWP.pdf

[20] Maryam Kamvar, Shumeet Baluja. A Large Scale Study of Wireless Search Behavior: Google Mobile Search. CHI-2006.

[21] Istvan Beszteri, Petri Vuorimaa. An XForms Based Solution for Adaptable Documents Editing. *SAC '05*, March 13-17, Santa Fe, New Mexico, USA, 2005.

[22] Jim Cai. Page Layout Adaptation for Small Form Factor Devices. http://www.cs.toronto.edu/~jcai/2514/term.doc

[23] T. Bickmore, B.N. Schillit, Digestor: Device-independent Access to the World Wide Web, 6th International World Wide Web Conference, Santa Clara, California, 1997.

[24] O. Buyukkokten, H. Garcia-Molina and A. Paepcke. Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones. Human-Computer Interaction Conference (CHI 2001), Seattle, Washington.

[25] Y. Chen, W. Ma, H. Zhang. Detecting Web page structure for adaptive viewing on small form factor devices. Proc. WWW' 03, pp. 225–233, May 2003.

[26] Buyukkokten, H. Garcia-Molina, A. Paepcke, and T. Winograd. Power Browser:

Efficient Web Browsing for PDAs. Proceedings of CHI, ACM Press, Amsterdam, 2000.

[27] S. Bjrk, L.E. Holmquist, J. Redstrm, I. Bretan, R. Danielsson, J. Karlgren, and K. Franzn. WEST: A Web Browser for small Terminals. ACM Sysmposium on User Interface Software and Technology, 1999.

[28] N. Milic-Frayling, R. Sommerer. Smartview: Enhanced document viewer for mobile devices. *Technical Report*, Microsoft Research, Cambridge, UK, November 2002.

[29] Lam H., Baudisch P. Summary thumbnails: readable overviews for small screen web browsers. CHI 2005, Portland, OR, pp. 681-690.

[30] User Agent Profile Approved Version 2.0. www.openmobilealliance.org/release_program/docs/UAProf/V2_0-20060206-A/OMA-TS-UAProf-V2_0-20060206-A.pdf