Capitolo 7

Sviluppi futuri

7.1 Generazione automatica di pagine WML

Con l'avvento della tecnologia WAP/WML abbiamo constatato la necessità di avere a disposizione uno strumento che consenta, così come effettuato per siti statici e dinamici, la generazione automatica di siti che supportano tale tecnologia. Prima di descrivere le problematiche che dovranno essere affrontate per estendere Penelope forniamo una breve descrizione di cos'è la tecnologia WAP/WML.

Il WAP (Wireless Application Protocol) nasce per consentire la comunicazione tra due tecnologie in rapidissima espansione: Internet e le reti mobili. Lo standard è stato promosso dal Wap forum, fondato nel 1997 con l'obiettivo di:

1. consentire la fruizione dei servizi disponibili su Internet tramite telefoni cellulari ed altri terminali wireless;
2. creare un protocollo di accesso in grado di funzionare con diverse tecnologie di rete mobile;
3. consentire la creazione di nuovi servizi distribuibili tramite diverse tecnologie di trasporto ed utilizzando differenti tipi di periferiche;
4. utilizzare gli standard esistenti dove possibile.

I siti che permettono l'interazione tra l'utente mobile e servizi internet devono essere sviluppati in uno specifico linguaggio, il WML.

Il WML (Wireless Markup Language) è un'applicazione XML sviluppata specificamente per distribuire contenuti e servizi su telefoni cellulari e più in generale su apparecchi che hanno le seguenti caratteristiche:

1. display di dimensioni limitate e con bassa risoluzione (la maggior parte degli attuali telefoni cellulari può visualizzare solo poche righe di testo e non più di 8-12 caratteri per riga);
2. limitate capacità di input (un telefono cellulare ha tipicamente una tastiera numerica e qualche tasto funzione);
3. limitate capacità di elaborazione perché dotati di Cpu a basso consumo e poca memoria;

Le caratteristiche del WML possono essere sintetizzate come segue:

- offre supporto per le immagini e per la formattazione del testo;
- si articola in *cards* a loro volta raggruppate in *deck* (un deck è simile ad una pagina html ed è identificato da una Url);
- offre strumenti per navigare tra card e deck ed include comandi per la gestione degli eventi, che possono essere utilizzati per navigare od eseguire degli script.

Occorre sottolineare che, poiché il WML è un'applicazione XML:

1. tutti i tag devono essere chiusi;
2. tutti i tag devono essere incapsulati (ad esempio, una sintassi del tipoparola è errata; la sintassi giusta è parola);
3. è necessario inserire un'intestazione che identifica il tipo di documento.

Dopo questa breve spiegazione della tecnologia cerchiamo di capire come dovrà operare Penelope.

Un'applicazione WML è organizzata in un insieme di *card* e *deck*. Le card specificano un frammento di interazione, ad esempio la scelta da un menu o il

riempimento di un campo. Le *card* sono raccolte in *deck*, ossia dei file con estensione "wml" che il server invia all'utente (analogamente ad una pagina html).

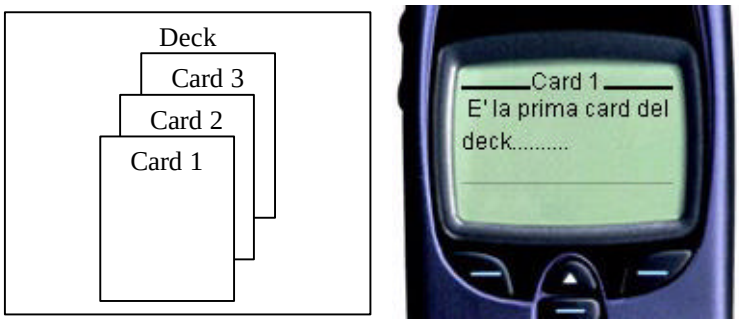


Figura 7.1: Schematizzazione *card* e d*eck*

Nell'esempio che segue illustriamo un semplice *deck* WML:

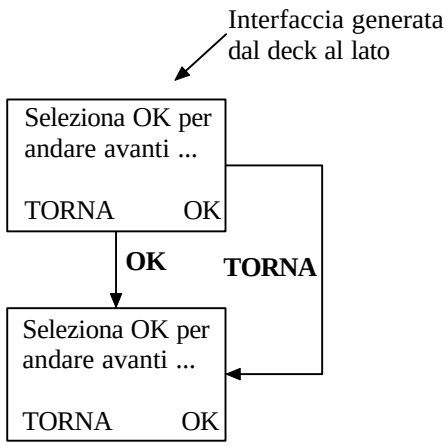
```
<?xml versione="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1/EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>

<card id="Card_1">
<do type="accept" label="OK">
<go href="#Card_2"/>
</do>
<p>
Seleziona "OK" per andare avanti...
</p>
</card>

<card id="Card_2">
<p>
Questa è la seconda Card...
</p>
</card>

</wml>
```



Penelope dovrà essere in grado, secondo una opportuna metodologia, di valutare:

1. nella generazione di un *deck*, come costruire le varie *Card* che lo compongono. Occorrerà molto probabilmente mantenere in memoria una pila di tag all'interno della quale ogni volta si apre un tag si effettua un push. Si immagini di aver generato un *deck* WML suddiviso in due *card* e si immagini che il contenuto informativo della *card* si riferisca ad un attributo composto LIST-OF di Penelope. Questo equivale a dire che alla fine della prima *card* occorrerà chiudere tutti i tag aperti e riaprirli immediatamente dopo nella seconda card. La pila di tag può risultare utile proprio per mantenere traccia di tutti i tag aperti.

2. nel caso di generazione automatica di pagine WML dinamiche, ad esempio JSP, deve essere in grado di riconoscere se la generica pagina è stata richiesta da un client HTTP oppure da un client WAP.

Durante lo svolgimento di questa Tesi si è affrontato questo ultimo problema e al fine di eliminare questa ambiguità nel riconoscimento del client si è implementata una servlet ("*dispatcherClient*") attraverso la quale viene riconosciuto il client connesso(mediante un controllo sull'*user-agent*) e viene di conseguenza redirezionato su un sito HTTP piuttosto che su un altro WAP o viceversa.

Si osservi, dalla figura 7.2, che la servlet "*dispatcherClient*" è centralizzata rispetto a tutte le pagine componenti il sito, ovvero ogni qual volta un client richiede una JSP per prima cosa viene invocata la servlet "*dispatcherClient*" che permetterà di determinare se la richiesta proviene da un client WAP oppure da un client HTTP.

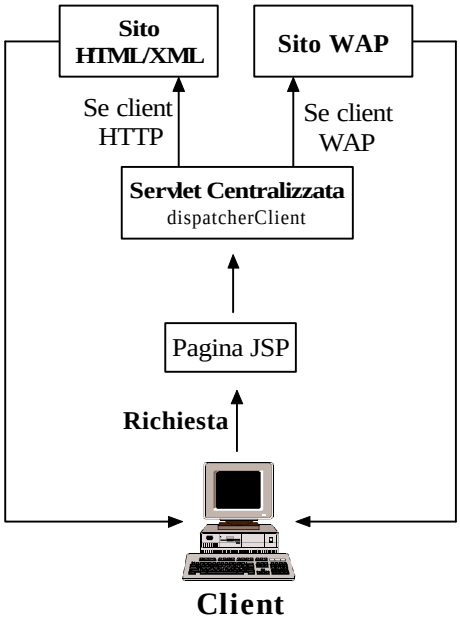


Figura 7.2: Architettura per il riconoscimento del client

Si riporta per completezza la servlet "*dispatcherClient*":

```
public class dispatcherClient extends HttpServlet
{

  private String IndexWml="http://vesuvio.dia.uniroma3.it/index.wml";
  private String IndexHtml="http://vesuvio.dia.uniroma3.it/index.html";

  PrintWriter out=null;

  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
    throws ServletException, IOException
  {
    Enumeration head=request.getHeaderNames();
        String name="user-agent";
        String $browser=request.getHeader(name);

      out = response.getWriter();
                  // set content type and other response header fields first
           //Si controlla se lo user-agent del client inizia con.....
           if ($browser.startsWith("Eric")|| // Ericsson WAP phones and emulators
        $browser.startsWith("Noki")|| // Nokia phones and emulators
        $browser.startsWith("WapI")|| // Ericsson WapIDE 2.0
        $browser.startsWith("MC21")|| // Ericsson MC218
        $browser.startsWith("AUR ")|| // Ericsson R320
```

```
    $browser.startsWith("R380")|| // Ericsson R380
    $browser.startsWith("UP.B")|| // UP.Browser
    $browser.startsWith("WinW")|| // WinWAP browser
    $browser.startsWith("UPG1")|| // UP.SDK 4.0
    $browser.startsWith("upsi")|| // another kind of UP.Browser ??
    $browser.startsWith("QWAP")|| // unknown QWAPPER browser
    $browser.startsWith("Jigs")|| // unknown JigSaw browser
    $browser.startsWith("Java")|| // unknown Java based browser
    $browser.startsWith("Alca")|| // unknown Alcatel-BE3 browser (UP based?)
    $browser.startsWith("MITS")|| // unknown Mitsubishi browser
    $browser.startsWith("MOT-")|| // unknown browser (UP based?)
    $browser.startsWith("My S")|| // unknown Ericsson devkit browser ?
    $browser.startsWith("WAPJ")|| // Virtual WAPJAG www.wapjag.de
    $browser.startsWith("fetc")|| // fetchpage.cgi Perl script from www.wapcab.de
    $browser.startsWith("ALAV")|| // yet another unknown UP based browser ?
    $browser.startsWith("Wapa"))  // another unknown browser (Web based "Wapalyzer"?)
    { // allora vuol dire che il client che deve essere servito è di tipo WAP
        String line=" ";
        BufferedReader   inStream=new BufferedReader(new
                                  InputStreamReader(new URL(IndexWml).openStream()));
        //allora viene scritto il dato di risposta e viene settato il contentType
        response.setContentType("text/vnd.wap.wml");
        while(line!=null){line=inStream.readLine();if (line!=null) out.println(line);}
          out.close();
    }
    // altrimenti vuol dire che il client che deve essere servito è di tipo HTTP
   else
    {
        String line=" ";
        BufferedReader   inStream2=new BufferedReader(new
                                    InputStreamReader(new URL(IndexHtml).openStream()));
        //allora viene scritto il dato di risposta e viene settato il contentType
        response.setContentType("text/html");
        while(line!=null){line=inStream2.readLine();if (line!=null) out.println(line);}
            out.close();
    }

    }// end Method doGet
}// end Class
```

7.2 Riscrittura del parser Penelope

Allo stato attuale il parser per il linguaggio Penelope è scritto in java, questo ovviamente crea degli inconvenienti ogni volta che deve essere modificata la sintassi, perché per inserire o modificare delle produzioni, occorre “navigare” all’interno di classi java e modificare a mano il codice sorgente.

Uno dei prossimi sviluppi previsto per Penelope è quello di riscrivere l’intero parser mediante JavaCC; JavaCC è un generatore di parser ed analizzatori lessicali in Java che utilizza la classe di grammatiche LL(1). Questa classe è ben nota per la sua semplicità ma generalmente non é utilizzata dai generatori di parser per la limitata espressività da cui é caratterizzata.

JavaCC si distingue per una serie di strumenti che consentono di disambiguare "localmente" le produzioni delle grammatiche LL(1) superandone così i limiti di espressività pur mantenendo intatta la loro semplicità di utilizzo.

7.3 Il sito parallelo per gli aggiornamenti

Accanto alla generazione di un sito web nasce anche l’esigenza di generare un sito che permetta l’aggiornamento online della base di dati sottostante.

Allo stato attuale gli aggiornamenti sulla base di dati, utilizzata per la generazione di un sito con Penelope, devono essere effettuati manualmente dall’amministratore direttamente sul su di essa, appare evidente la necessità di fornire, anche all’utente non addetto ai lavori, un sito che attraverso delle form dinamiche, che rispecchiano la struttura del sito pubblico (sito navigabile dall’utente), permettano un rapido aggiornamento della base di dati.

In questo modo l’amministratore del sito può interagire con le form attraverso un paradigma navigazionale che rispecchia quello del sito pubblico.

Occorre definire pertanto un modello per le form che permetta:

1. di catturare i vincoli della base di dati;
2. di definire un algoritmo per la generazione dinamica delle form.

Occorre tuttavia notare che particolare attenzione andrà riservata alle cancellazioni:

- Se la cancellazione riguarda gli attributi della funzione URL, si dovrà procedere anche alla rimozione della pagina il cui URL era calcolato sulla base dei suoi attributi;
- Si supponga inoltre di voler cancellare il valore di un campo di una generica tabella Ti, occorrerà rimuovere anche tutti gli elementi delle tabelle ad essa relazionate; operazione non sempre agevole e realizzabile in modo automatico senza una interazione complessa con l'utente.