Capitolo 5

Ulteriori estensioni del nuovo prototipo

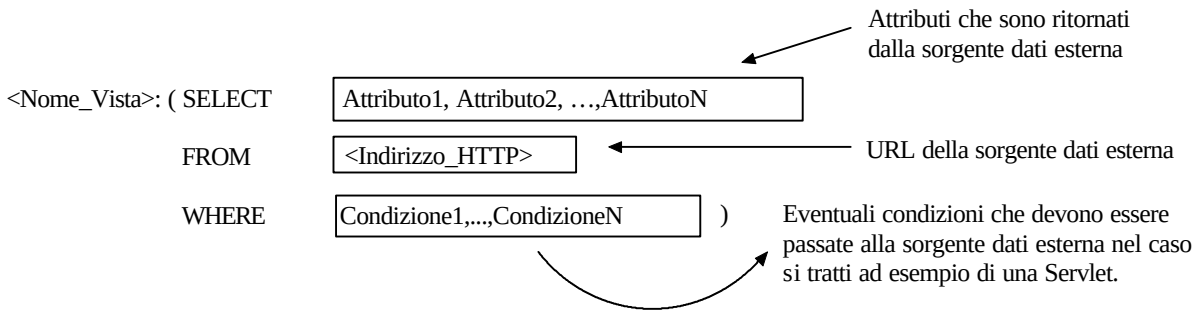
In questo capitolo verranno descritte ulteriore estensioni sviluppate sul prototipo Penelope, si descriverà il pattern, DBConnectionManager, utilizzato per gestire pool di connessione ad uno o più basi di dati e lo sviluppo di una semplice interfaccia user-friendly per l'utente, la PenelopeGUI.

5.1 Dati provenienti da sorgenti esterne

Abbiamo visto nel Capitolo 4 che Penelope genera siti a partire da una base di dati relazionale sottostante. Il nuovo prototipo sviluppato è stato esteso in modo che sia possibile prendere i dati anche da una qualsiasi sorgente esterna, ad esempio Servlet, file,…, disponibile sul web.

Per chiarire meglio il concetto si immagini di voler utilizzare non una tabella proveniente dalla base di dati bensì una tabella proveniente da una Servlet che si trova in un qualsiasi indirizzo web. Questo è possibile con Penelope, anche se la sorgente esterna non può restituire informazioni in un formato qualsiasi ma deve attenersi a codificare i dati in un opportuno formato di scambio(XML) che andremo a descrivere.

Per fare in modo che Penelope gestisca tabelle provenienti da sorgenti dati esterne, la sintassi da utilizzare, nella clausola USING, è la seguente:



La tabella ritornata deve essere codificata secondo il seguente formato di scambio XML:

```
<table>
<tuple><Attribute1>Value1</Attribute1>.....<AttributeN>ValueN</AttributeN></tuple>
<tuple><Attribute1>Value1</Attribute1>.....<AttributeN>ValueN</AttributeN></tuple>
.....
<tuple><Attribute1>Value1</Attribute1>.....<AttributeN>ValueN</AttributeN></tuple>
</table>
```

Gli elementi <table> e <tuple> identificano rispettivamente la tabella ed una tupla. L'elemento <AttributeX> permette di determinare l'attributo ritornato.

ESEMPIO

Consideriamo una base di dati, nella quale sia presenta la tabella relazionale Professore, e supponiamo che esista una servlet che consenta di interrogare tale base di dati.

Cl	Cognome	Nome	email	Stanza	Tel
Informatica	Di Battista	Giuseppe	gdb@dia...	x	06....
Informatica	Atzeni	Paolo	atzeni@dia..	y	06....
Elettronica	Neri	Alessandro	neri@ele...	z	06....
Meccanica	Cerri	Giovanni	cerri@....	w	06....
Elettronica	Assanto	Gaetano	assanto@ele..	k	06....
Informatica	Cialdea	Marta	cialdea@...	j	06....

Figura 5.1: Tabella relazionale Professore

Immaginiamo che la servlet esegua la seguente query:

```
SELECT Cl, Cognome, Nome, email
FROM Professore
```

Il risultato che si otterrà eseguendo la query sarà il seguente cursore :

Cl	Cognome	Nome	email
Informatica	Di Battista	Giuseppe	gdb@dia...
Informatica	Atzeni	Paolo	atzeni@dia..
Elettronica	Neri	Alessandro	neri@ele...
Meccanica	Cerri	Giovanni	cerri@....
Elettronica	Assanto	Gaetano	assanto@ele..
Informatica	Cialdea	Marta	cialdea@...

Figura 5.2: Cursore ottenuto eseguendo la query

Ora la servlet quando invocata dovrà semplicemente restituire un *InputStream* così codificato:

```
<table>
<tuple><Cl>Informatica</Cl><Cognome>Di Battista</Cognome><Nome>Giuseppe</Nome><email>gdb@dia.uniroma3.it</email></tuple>
<tuple><Cl>Informatica</Cl><Cognome>Atzeni</Cognome><Nome>Paolo</Nome><email>atzeni@dia.uniroma3.it</email></tuple>
<tuple><Cl>Elettronica</Cl><Cognome>Neri</Cognome><Nome>Alessandro</Nome><email>neri@ele.uniroma3.it</email></tuple>
<tuple><Cl>Meccanica</Cl><Cognome>Cerri</Cognome><Nome>bo</Nome><email>cerri@mec.uniroma3.it</email></tuple>
<tuple><Cl>Elettronica</Cl><Cognome>Assanto</Cognome><Nome>bo1</Nome><email>assanto@ele.uniroma3.it</email></tuple>
<tuple><Cl>Informatica</Cl><Cognome>Cialdea</Cognome><Nome>Marta</Nome><email>cialdea@dia.uniroma3.it</email></tuple>
</table>
```

Figura 5.3: *InputStream* restituito dalla servlet

Lo stesso discorso vale se invece di avere una servlet si ha direttamente un file contenente il risultato. Se ad esempio il file si chiama "Query_Prof.txt" e restituisce *l'InputStream* di figura 5.3 allora attraverso il seguente page-scheme:

```
SCHEME file:/c:sites\dept2\
ON CourseList

DEFINE PAGE Dynamic EducationPage : "Education"
STYLE sites\styles\verticalXML.STY(Our Courses)
AS URL("indexEducation");

   ListaCorsodilaurea: LIST-OF (    Corsolaurea : TEXT = <CL.Cl>;
                              ListaProfessori: LIST-OF (  Cognome : TEXT = <Cognome>;
                                                          nome : TEXT = <Nome>;
                                                          email : TEXT = <email>;
                                                       ListaCorsi : LIST-OF (
                                                             CourseNome : TEXT = <CorsoNome>;
                                                                          );
                                                    );
                           );
                                                                          Chiamata
                                                                          esterna
USING    CL,
         Prof : (SELECT Cl, Cognome, Nome, email
                 FROM  http://vesuvio.dia.uniroma3.it/examples/Query_Prof.txt),


         ProfCourse : (SELECT  Cognome, CorsoNome
                       FROM Professore, Corsi
                       WHERE Professore.Cognome=Corsi.Docente),

ORDER BY CorsoNome

END
```

Si otterrà la seguente pagina JSP:

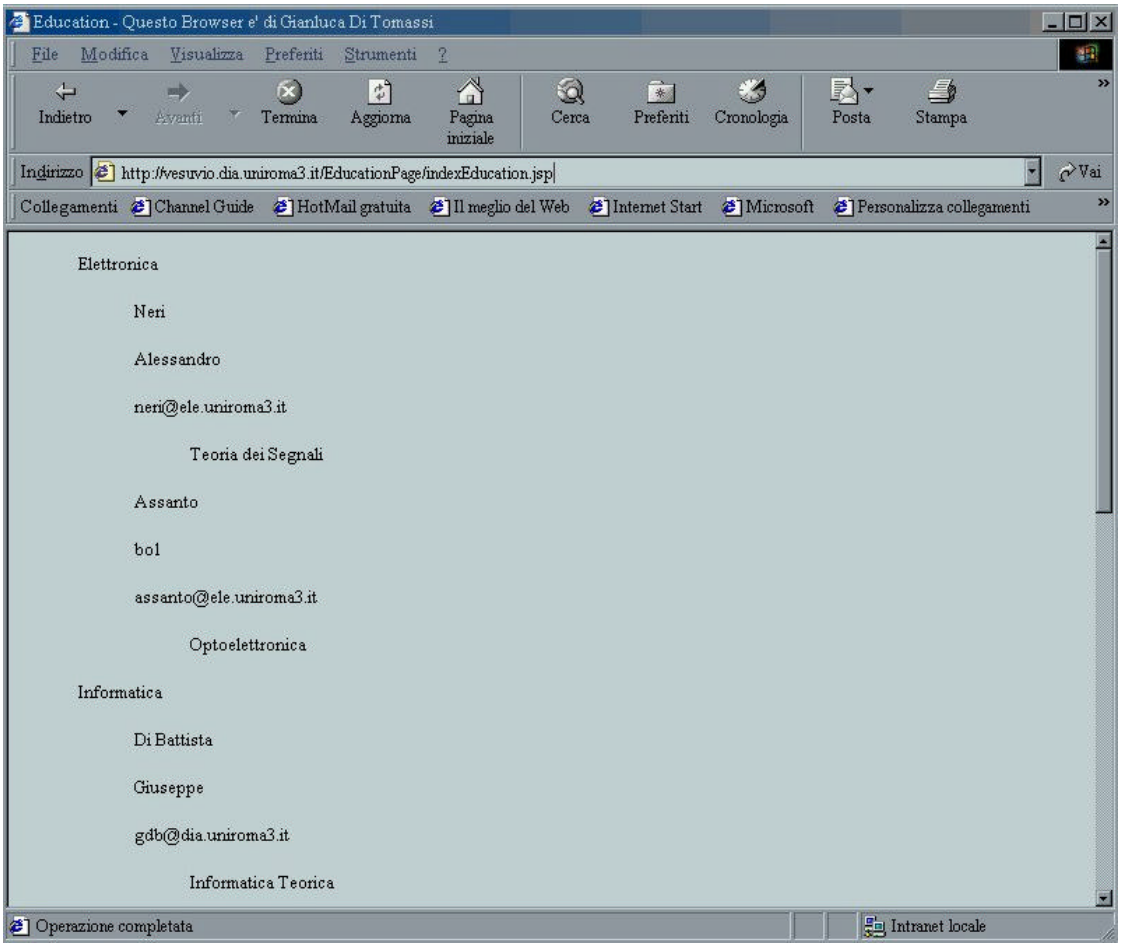


Figura 5.4: Risultato ottenuto invocando la pagina “indexEducation.jsp”

5.2 L’interfaccia `ResultSetPenelope` e una sua implementazione

Al fine di poter gestire in maniera agevole lo stream ritornato dalla sorgente dati esterna, è stata definita l’interfaccia:

```
interface ResultSetPenelope {
  String getString(String key);
  boolean next(Properties lista);
  void beforeFirst();
}
```

Una sua implementazione, "ResultSetExternal", è stata realizzata nel corso di questa tesi per realizzare pagine JSP che utilizzino sorgenti dati esterne.

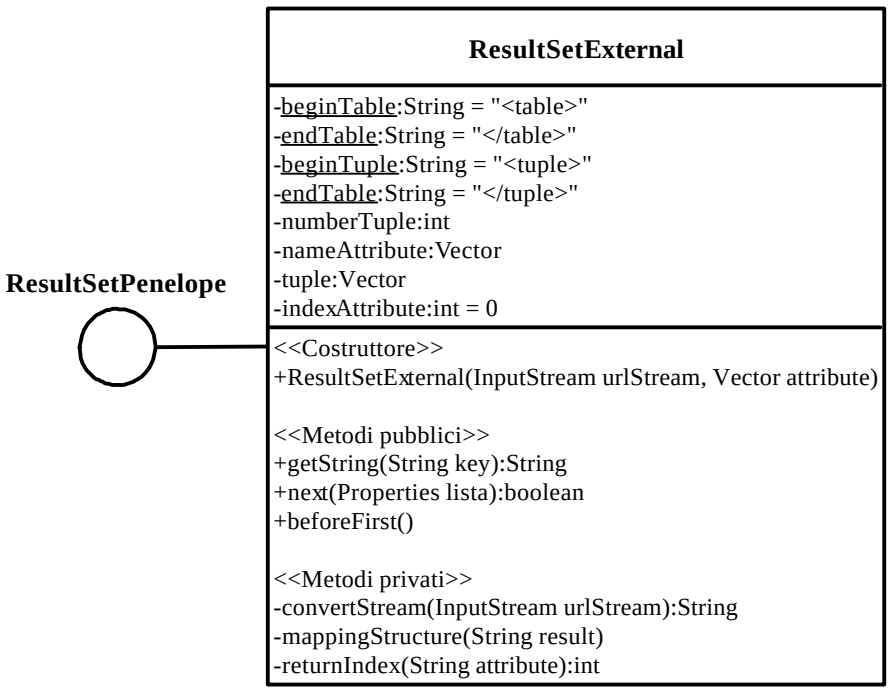


Figura 5.5: Diagramma dell'interfaccia e della sua implementazione

Illustriamo di seguito, sommariamente, gli attributi e la funzione svolta dai singoli metodi della classe ResultSetExternal:

Attributi della classe ResultSetExternal

- *beginTable* Tag di apertura identificativo della tabella;
- *endTable* Tag di chiusura identificativo della tabella;
- *beginTuple* Tag di apertura identificativo della tupla;
- *endTuple* Tag di chiusura identificativo della tupla;
- *numberTuple* variabile intera che tiene conto di quale tupla si sta analizzando;
- *nameAttribute* Vettore contenente i nomi degli attributi;
- *tuple* Vettore delle tuple i cui elementi sono ancora dei vettori contenenti i valori degli attributi;

- *indexAttribute* indice dell'attributo.

Metodi della classe ResultSetExternal

- ResultSetExternal(InputStream urlStream, Vector attribute)

  Costruttore della classe prende in input un *InputStream*, che è la tabella ritornata dalla nostra sorgente dati esterna, e il vettore degli attributi che sono presenti in ogni tupla. Nel costruttore viene inizializzata la *numberTuple* a –1 e vengono invocati i metodi *convertStream* (per la conversione dell'*InputStream* in stringa) e *mappingStructure* (per costruire in memoria una struttura dati rappresentante la nostra tabella);

- ConvertStream(InputStream urlStream)

  Consente di convertire uno stream di input in una stringa;

- mappingStructure(String input)

  Metodo che si occupa di effettuare il mapping dello *stream* ritornato dalla servlet in un vettore di vettori. In ingresso viene passata la stringa "input" che non e' altro che lo *stream* ritornato dalla servlet. La stringa come già detto è codificata secondo la seguente sintassi:

```
<table>
<tuple><Attribute1>Value1</Attribute1>.....<AttributeN>ValueN</AttributeN></tuple>
<tuple><Attribute1>Value1</Attribute1>.....<AttributeN>ValueN</AttributeN></tuple>
.....
<tuple><Attribute1>Value1</Attribute1>.....<AttributeN>ValueN</AttributeN></tuple>
</table>
```

  Questa viene mappata nella struttura 'tuple' che è un Vector di Vector :

  [[Valore1, Valore2,....,ValoreN],
  [Valore1, Valore2,....,ValoreN],
  .....,
  [Valore1, Valore2,....,ValoreN] **]**

Il vettore esterno rappresenta la tabella mentre quelli interni sono le sue tuple. Ogni elemento del vettore delle tuple non è altro che il valore di un attributo.

Parallelamente viene anche inizializzato il vettore *nameAttribute* che sarà quindi così costituito:

[Attributo1, Attributo2,....,AttributoN]

In questo modo le due strutture create sono tra loro in corrispondenza biunivoca e sarà possibile in ogni momento determinare per un qualsiasi attributo quale sia il suo valore corrispondenete alla generica tupla *i-esima*.

ESEMPIO

Immaginiamo che la tabella ritornata dalla sorgente dati esterna sia la seguente:

```
<table>
<tuple><Cl>Informatica</Cl><Cognome>Di Battista</Cognome><Nome>Giuseppe</Nome></tuple>
<tuple><Cl>Informatica</Cl><Cognome>Atzeni</Cognome><Nome>Paolo</Nome></tuple>
<tuple><Cl>Elettronica</Cl><Cognome>Neri</Cognome><Nome>Alessandro</Nome></tuple>
<tuple><Cl>Meccanica</Cl><Cognome>Cerri</Cognome><Nome>Giovanni</Nome></tuple>
<tuple><Cl>Elettronica</Cl><Cognome>Assanto</Cognome><Nome>Gaetano</Nome></tuple>
<tuple><Cl>Informatica</Cl><Cognome>Cialdea</Cognome><Nome>Marta</Nome></tuple>
</table>
```

La struttura che viene costruita in memoria sarà:

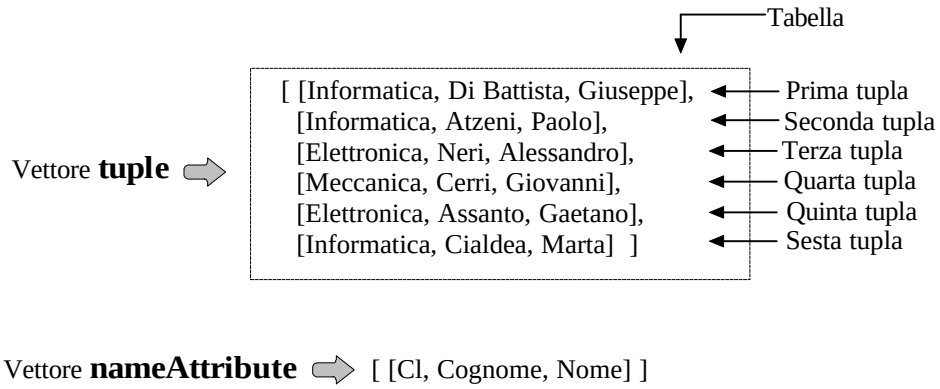


Figura 5.6: Rappresentazione in memoria di una tabella

Per determinare il valore assunto dall'attributo "Cognome" nella quarta tupla, quindi *numberTuple*=4, quello che basterà fare è :

1. Determinare la posizione dell'attributo nel vettore *nameAttribute* (1);

2. Accedere all'elemento *numberTuple* di *tuple* per posizionarsi sulla tupla desiderata(4), quindi accedere, nella tupla, all'elemento che si trova nella posizione determinata nel punto 1.(1).

- returnIndex(String attribute)

  Ritorna la posizione occupata dall'attributo '*attribute*' nel vettore *nameAttribute*;

- beforeFirst()

  Riporta il contatore di tuple, '*numberTuple'*, al valore –1;

- Next(Properties condition)

  Incrementa *numberTuple* si posiziona sulla tupla successiva scorrendo ad una ad una tutte le tuple finchè non raggiunge la fine o trova una tupla che soddisfa le condizioni elencate nell'oggetto condition (si incarica quindi di effettuare la selezione sulle condizioni che gli vengono passate in input);

- GetString(String key)

  Prende in input il nome dell'attributo e restituisce il suo valore;

5.2.1 Traduzione in JSP di dati provenienti da sorgenti esterne

Al fine di descrivere agevolmente come venga tradotto da Penelope, il caso in cui una o più tabelle definite nella clausola USING provengano da sorgenti dati esterne, in termini di codice JSP riferiamoci al seguente esempio:

ESEMPIO

Consideriamo il seguente frammento di PL:

```
ON CourseList

DEFINE PAGE Dynamic EducationPage : "Education"
AS URL("indexEducation");
  ........
            ListaProfessori: LIST-OF (  Cognome : TEXT = <Cognome>;
                                        nome : TEXT = <Nome>;
                                        email : TEXT = <email>;
                                      );
  ..........


USING  ...,
       Prof : (SELECT Cl, Cognome, Nome, email
               FROM http://vesuvio.dia.uniroma3.it/examples/Servlet_Prof),
       .........
END
```

Di seguito riportiamo la traduzione Penelope in termini di pagina JSP:

```
1.  <%@ page import="ResultSetExternal" %>

    ..................
    <%
2.      boolean executeListaProfessori = false;
3.      ResultSetExternal ListaProfessori = null;
    %>
  ..................

  <UL>
  <%
4.      Vector attributiListaProfessori = new Vector();
5.      if (!executeListaProfessori)
        {
         try
         {
6.         URL urlListaProfessori = new URL("http://vesuvio.dia.uniroma3.it/examples/Servlet_Prof");
7.         executeListaProfessori = true;
8.         attributiListaProfessori.addElement("Cl");
9.         attributiListaProfessori.addElement("Cognome");
10.        attributiListaProfessori.addElement("Nome");
11.        attributiListaProfessori.addElement("email");
12.        ListaProfessori = new ResultSetExternal(urlListaProfessori.openStream(), attributiListaProfessori);
         }catch(MalformedURLException e) {}
        }
13.   Properties listaListaProfessori = new Properties();
14.   listaListaProfessori.setProperty("Cl",checkString(ListaCorsodilaureaCl));
15    ListaProfessori.beforeFirst();
16.   while (ListaProfessori.next(listaListaProfessori))
        {
17.      String ListaProfessoriCl = ListaProfessori.getString("Cl");
18.      String ListaProfessoriCognome = ListaProfessori.getString("Cognome");
19.      String ListaProfessoriNome = ListaProfessori.getString("Nome");
20.      String ListaProfessoriemail = ListaProfessori.getString("email");
    %>
```

```
21.      <P><% if (ListaProfessoriCognome!= null) out.print(ListaProfessoriCognome); %></P>
22.      <P><% if (ListaProfessoriNome!= null) out.print(ListaProfessoriNome); %></P>
23.      <P><% if (ListaProfessoriemail!= null) out.print(ListaProfessoriemail); %></P>
24.  <%  }/*end while ListaProfessori*/  %>

</UL>
```

Spieghiamo per punti quanto generato. Si importa la classe ResultSetExternal (punto 1).Si dichiara un flag booleano “executeListaProfessori” che viene utilizzato per determinare se è già stata invocata la sorgente dati esterna(true) o meno(false) (punto 2). Si inizializza a null il nostro oggetto di tipo ResultSetExternal “ListaProfessori” (punto 3).

Si dichiara un vettore, “attributiListaProfessori”(punto 4), i cui elementi sono gli attributi ritornati dalla sorgente dati esterna. Si Controlla che la servlet non sia già stata invocata (punto 5). Se la sorgente non è mai stata invocata allora si crea un oggetto di tipo URL specificando il suo l’indirizzo http (punto 6), si pone a “true” il flag “executeListaProfessori” (punto 7), si inseriscono i nomi degli attributi nel vettore “attributiListaProfessori” (punti 8,910,11.) e infine si istanzia il ResultSetExternal “ListaProfessori” invocando il costruttore della classe, passandogli l’*InputStream* ritornato dalla servlet e il vettore “attributiListaProfessori” (punto 12.).

Si dichiara una variabile “listaListaProfessori” che contiene le condizioni di join tra il livello di nidificazione attuale e quello precedente (punto 13.). Si inseriscono tali condizioni(14.). Ci si posiziona prima della prima tupla (punto 15.).

Si esegue un ciclo finché si hanno tuple (punto 16.) e si istanziano “n” variabili stringa quanti sono gli attributi ritornati (punto 17,18,19,20.). Se non sono nulli si stampano gli attributi “ListaProfessoriCognome”, “ListaProfessoriNomi”, “ListaProfessoriemail” (punti 21,22,23).

Infine si termina il ciclo while (punto 24)

5.3 Architettura del pattern DBConnectionManager

Lo studio e l'implementazione di tale pattern si è reso necessario, al fine di poter gestire pool di connessioni verso *n* basi di dati.

L'architettura del pattern DBConnectionManager può essere schematizzata attraverso il seguente diagramma delle classi.

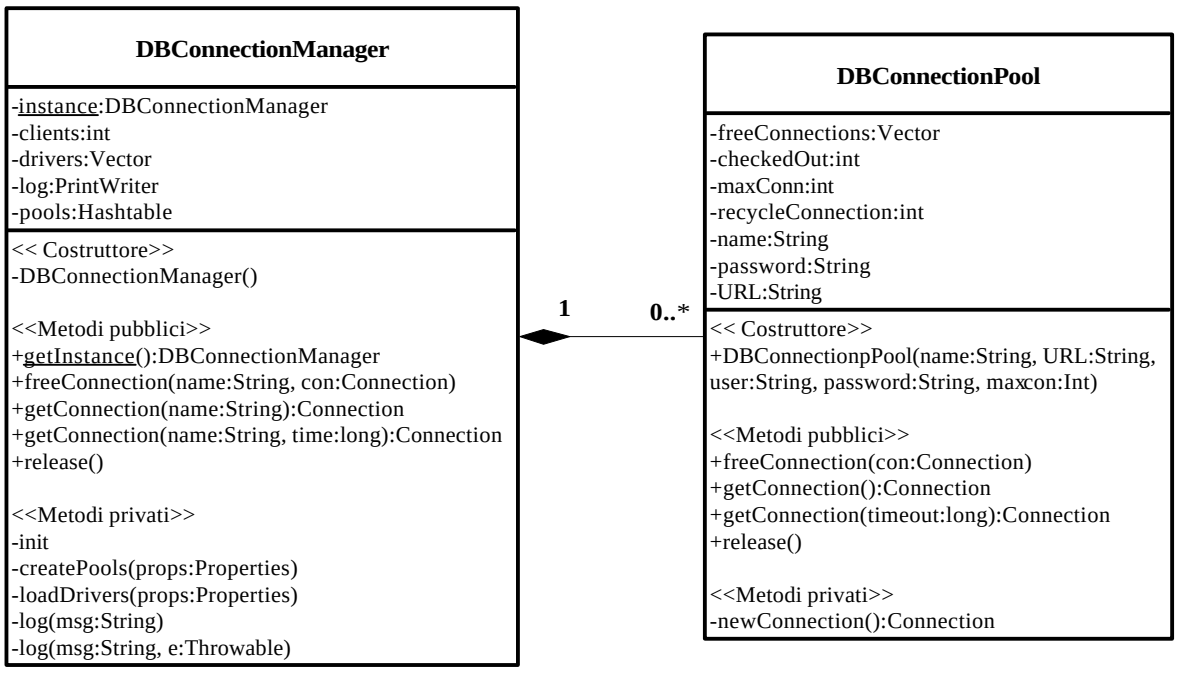


Figura 5.7: Architettura del pattern DBConnectionManager

Dall'analisi della struttura interna della classe DBConnectionManager, si può notare che si tratta di un "singleton" ("pattern" appartenente alla cosiddetta categoria "creational"). Esso è utilizzato ogni qual volta si voglia essere certi che, in ogni istante di tempo, esista una sola istanza di una determinata classe. Caso tipico delle classi manager (appunto) o comunque di quelle che si occupano di centralizzare l'accesso a determinate risorse. A tal fine i metodi costruttori sono dichiarati privati. La logica conseguenza è che un oggetto della classe possa essere creato solo dal suo interno sotto lo stretto controllo di un opportuno metodo; in questo caso *getIstance*().

Si tratta di un metodo statico (e quindi riportato sottolineato), che restituisce sempre lo stesso riferimento al medesimo oggetto. A tal fine è necessario disporre anche di un attributo interno statico che contenga il riferimento a sé stesso (nel nostro caso *instance*). Pertanto a seguito dell'invocazione del metodo *getIstance()*, non si fa altro che restituirne il valore all'esterno. L'unica eccezione è costituita dalla prima invocazione; in tal caso la classe deve occuparsi di inizializzarne il valore con quello restituito dal costruttore privato.

All'interno dell'elenco dei metodi delle classi, sono stati introdotti tre stereotipi:

"<<Construttore>>" , "<<Metodi pubblici>>" e <<Metodi privati>> al fine di ordinare la relativa lista per funzionalità.

Dall'analisi delle molteplicità (0..n) si può osservare che la classe DBConnectionPool è in grado di gestire un "pool" di risorse per ciascuna tipologia di connessione (più "database server"), mentre ciascuna istanza della classe DBConnectionManager può essere gestita da una sola classe DBConnectionPool

Naturalmente, la classe DBConnectionPool incapsula al proprio interno la connessione fisica con il "database server". Le relative istanze vengono create solo quando si genera una reale esigenza: quando non vi é alcuna connessione disponibile ed il numero massimo di istanze previste non è stato ancora raggiunto.

La classe DBConnectionManager identifica una opportuna basi di dati per mezzo del metodo *getConnection* al quale viene passato come parametro il nome della sorgente dati ODBC/JDBC rispetto alla quale si vuole creare un pool di connessioni, ne contiene le informazioni necessarie, però non realizza alcuna connessione fisica con il "database server". La connessione fisica avviene sempre nel metodo *getConnection* accoppiando un oggetto della classe DBConnectionManager ad un'opportuna istanza della classe DBConnectionPool, come si può ben notare dalla molteplicità.

Quando viene richiesta una nuova connessione, la classe DBConnectionPool verifica la presenza di una connessione già esistente ed inutilizzata, in caso affermativo la alloca, in caso negativo verifica la possibilità di istanziarne una nuova.

Il client accede unicamente alla classe DBConnectionManager (si tratta di una classica architettura a due strati), quando richiede l'allocazione di una risorsa di connessione invoca (implicitamente) un metodo *getConnection* della classe DBConnectionPool, mentre quando ne ha terminato l'utilizzo richiama il metodo *release*. Occorre tuttavia evidenziare che la gestione del rilascio delle connessioni viene gestito in maniera molto efficiente; si utilizza un vettore *freeConnections*, i cui elementi sono le connessioni che devono essere rilasciate, come una pila.

In questo modo, senza dover ogni volta rilasciare e istanziare connessioni, quando un nuovo client ne richiede una si verifica se il vettore *freeConnections* contiene uno o più elementi, in caso affermativo il primo elemento(che non è altro che una connessione) viene assegnato al client e rimosso dalla pila, in caso negativo occorre necessariamente istanziare una nuova connessione.

Quando viene invocato il metodo *release* vengono liberate tutte le connessioni disponibili, cioè vengono rimossi tutti gli elementi del vettore *freeConnections*.

5.3.1 Il file *db.properties*

Le informazioni relative ai vari pool di connessioni che si vogliono gestire, insieme ad altri parametri, sono presi da un file di *properties* che deve essere precedentemente editato. Il file ha la seguente struttura:

```
logfile=<PATH_LOG>

drivers=<DRIVERS>

<Pool_Name>.url=<URL_Sorgente_Dati>
<Pool_Name>.maxconn= Intero
<Pool_Name>.recycleConnection= Intero
<Pool_Name>.user= Stringa
<Pool_Name>.password= Stringa

.....
```

Figura 5.8: Struttura del file "*db.properties*"

Quindi è possibile per l'utente specificare il path dove si vuole memorizzare il file

di log (opzionale), i "drivers" che possono essere utilizzati e poi tutte le informazioni relative ai singoli "pool" di connessione che si vogliono definire. Ovviamente si possono definire un numero qualsiasi di pool. Riportiamo nella figura 5.9 un esempio di file di properties.

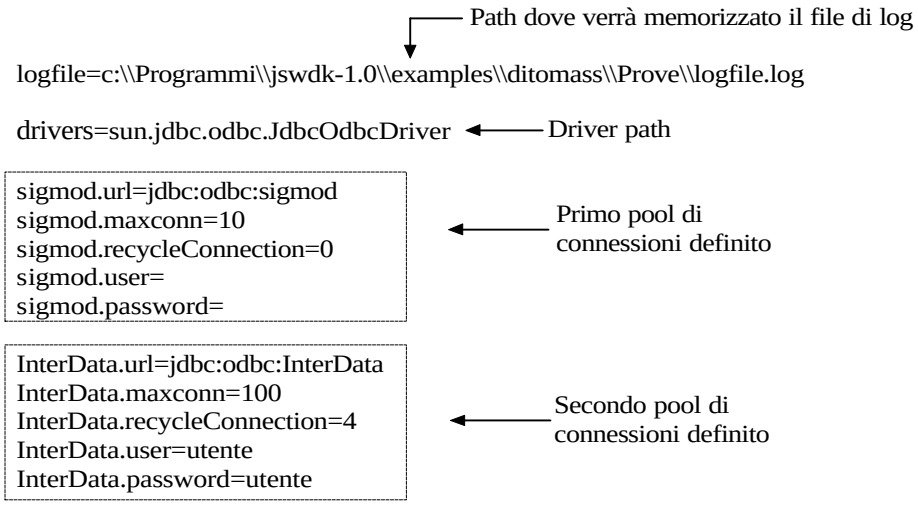


Figura 5.9: Esempio di file di properties

5.3.2 Attributi e metodi del pattern

Illustriamo di seguito, sommariamente, gli attributi e la funzione svolta dai singoli metodi delle le due classi componenti il pattern:

Attributi della classe DBConnectionManager

- *instance* è un oggetto di tipo DBConnectionManager e ne rappresenta l'unica istanza;
- *clients* contatore dei client collegati;
- *drivers* vettore contenente tutti i drivers specificati nel file di "properties";
- *log* oggetto di tipo *PrintWriter* utilizzato per scrivere il file di log;
- *pools* Hashtable contenente i pool di connessioni che si vogliono attivare e che vengono letti dal file di "properties" (contiene (coppie poolName,pool)).

Attributi della classe DBConnectionPool

- *checkedOu*t intero che tiene conto del numero di connessioni che sono state assegnate;
- *freeConnections* vettore contenente le connessioni che sono state rilasciate ma non ancora rimosse e che quindi possono essere riassegnate ad un client;
- *maxConn* massimo numero di connessioni sostenibili per un determinato pool;
- *recycleConnection* intero rappresentante il numero di connessioni (più una) che si vogliono utilizzare per un singolo pool; una volta raggiunto tale numero le connessioni vengono riutilizzate da altri client. Ad esempio se per il pool "X" *recycleConnection*=3 vuol dire che, mentre ai primi tre client vengono assegnate quattro connessioni distinte, dal quarto client in poi appena visualizzata la pagina richiesta viene rilasciata la connessione e resa disponibile per il successivo client. In questo caso con quattro connessioni si gestiscono tutti i client.
- *name* nome del pool;
- *user* eventuale username richiesta per accedere alla base di dati;
- *password* presente solo nel caso sia presente una username per accedere alla base di dati;
- *URL* rappresenta l'URL JDBC per la base di dati.

Metodi della classe DBConnectionManager

- *getInstance:* Controlla se l'oggetto *instance* è nullo, in caso affermativo crea una nuova istanza della DBConnectionManager, incrementa il numero di clienti (variabile *clients*) e ritorna *instance* altrimenti non ne istanza di nuove;
- *freeConnection:* Istanzia un oggetto DBConnectionPool che si occupa di richiamare il metodo *freeConnection* che accoda l'oggetto *Connection* nel vettore freeConnections, dopodiché sottrae uno al contatore dei client (variabile client);
- *getConnection:* Ritorna una connessione aperta. Se non c'è ne una disponibile e il numero max di connessioni non è stato raggiunto, ne viene creata una nuova;
- *release:* chiude tutte le connessioni aperte e rilascia tutti i "drivers";

- *createPools:* Crea una istanza della DBConnectionManager basandosi sulle informazioni contenute nel file di properties. Una DBConnectionManager può essere definita con le seguenti proprietà:

<poolname>.url	JDBC URL per la base di dati
<poolname>.user	Username per la base di dati (opzionale)
<poolname>.password	Password per la base di dati (solo se presente user)
<poolname>.maxconn	Massimo numero di connessioni (opzionale)
<poolname>.recycleConnection	Quando iniziare a "riciclare" le connessioni

- *init:* Carica il file di properties (db.properties) e istanzia un oggetto di tipo *properties* poi invoca i metodi *loadDrivers* e *createPools*, ai quali viene passato l'oggetto p*roperties* e che servono rispettivamente per caricare e registrare tutti i driver JDBC, il primo, per creare un pool di connessioni ,il secondo;
- *loadDrivers:* Carica e registra tutti i "drivers" JDBC;
- *log:* Scrive un messaggio sul file di log.

Metodi della classe DBConnectionPool

La DBConnectionPool è una classe interna alla DBConnectionManager; si occupa di gestire realmente il pool di connessioni. I suoi metodi, tutti privati ed invocati dalla DBConnectionManager sono:

- *getConnection:* Verifica se è già presente una connessione dal pool. Se non c'è ne una disponibile viene creata solo se non è stato raggiunto il numero massimo di connessioni;
- *newConnection:* Metodo invocato per creare una nuova connessione a sua volta invoca la *getConnection*;
- *release:* chiude tutte le connessioni che sono state rilasciate e quindi rese disponibili nel vettore *freeConnections*.

5.4 La Penelope GUI

Nell'ambito di questa tesi è stata sviluppata una shell grafica che fornisce un'interfaccia "user-friendly" per l'uso del prototipo Penelope.
Tale shell è mostrata in Figura 5.10.

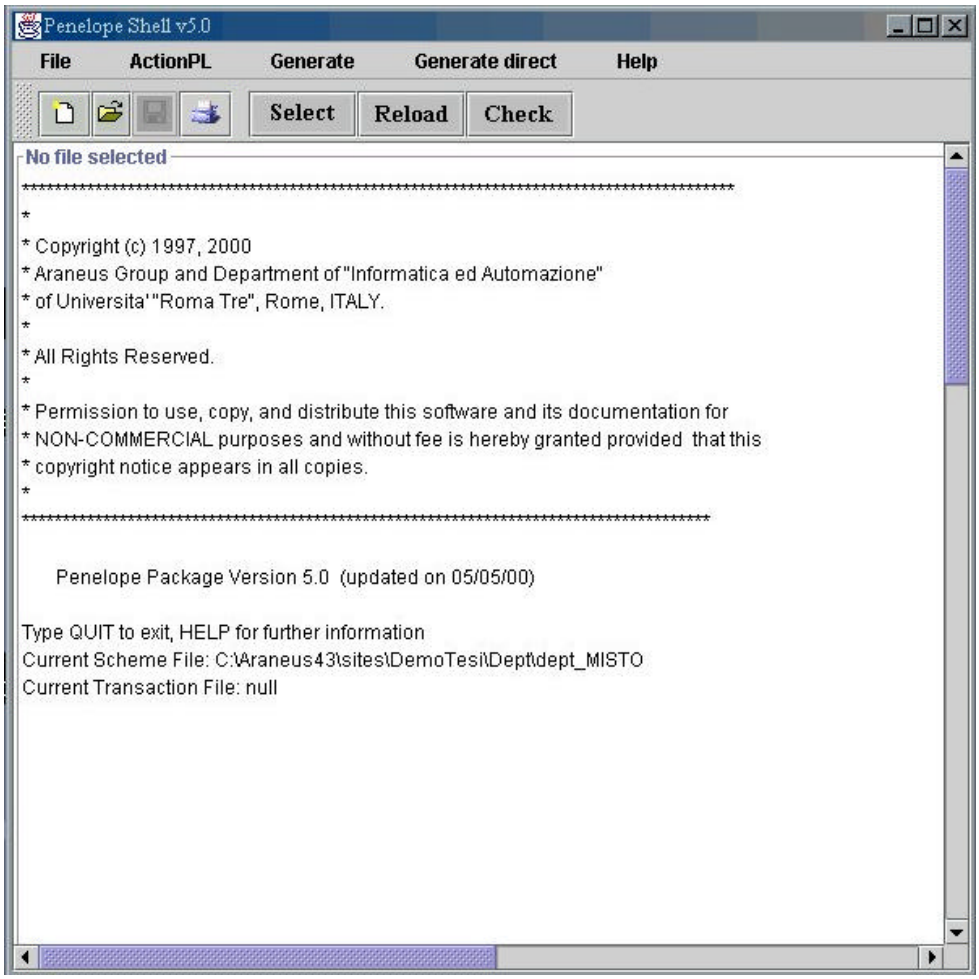


Figura 5.10: La PenelopeGUI.

Come si può notare, la shell è dotata di una barra di menu e di una serie di tasti che permettono l'esecuzione dei comandi.

Il primo menu (Figura 5.12) permette di editare, aprire, salvare e stampare un file, pertanto la PenelopeGUI può essere utilizzata anche come editor testuale per editare i

propri PL. E' inoltre possibile, sempre dal menu file attraverso "*Manage DB*" *e* "*Setup ODBC DataSources*", gestire sia la base di dati utilizzata nel PL sia inserire una nuova sorgente dati ODBC. Se si vogliono attivare queste due voci del menu accorre che sia presente il file "*PenelopeGUI.properties*" all'interno del quale viene specificato quale DBMS si utilizza per gestire la base di dati e quale eseguibile attiva l'ODBC.

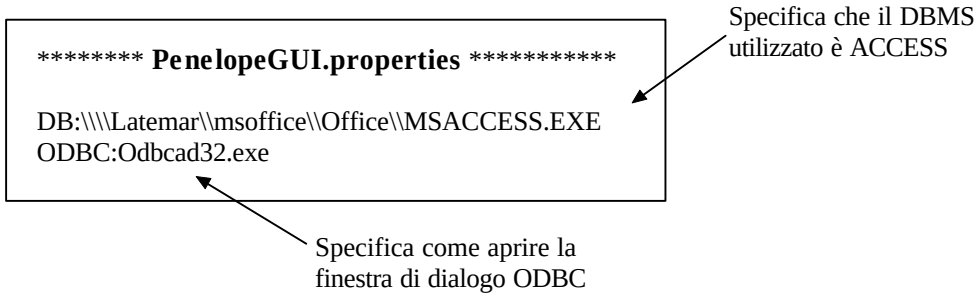


Figura 5.11: Il file di configurazione "*PenelopeGUI.properties*"

Quando si vuole gestire la base di dati ("*Manage DB*") viene aperta una finestra di dialogo che permette di selezionare e di aprire la base di dati scelta con il DBMS specificato nel file di properties.

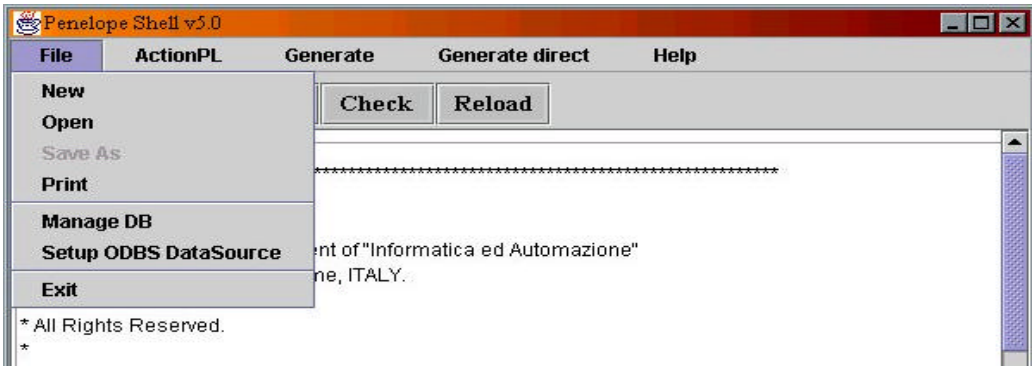


Figura 5.12: Menu "*File*" della PenelopeGUI

Il secondo menu(Figura 5.13) permette di selezionare tutte le operazioni disponibili su un PL, cioè permette di selezionare un PL, di ricaricarlo (operazione necessaria

quando si modifica il PL corrente) e di effettuare il check, cioè di controllarne la correttezza semantica delle istruzioni con riferimento allo schema della base di dati.

Figura 5.13:Menu "*ActionPL*" della PenelopeGUI

Quando si seleziona un PL viene visualizzata una finestra di dialogo (vedi Figura 5.14) che permette di selezionare il file d'interesse.

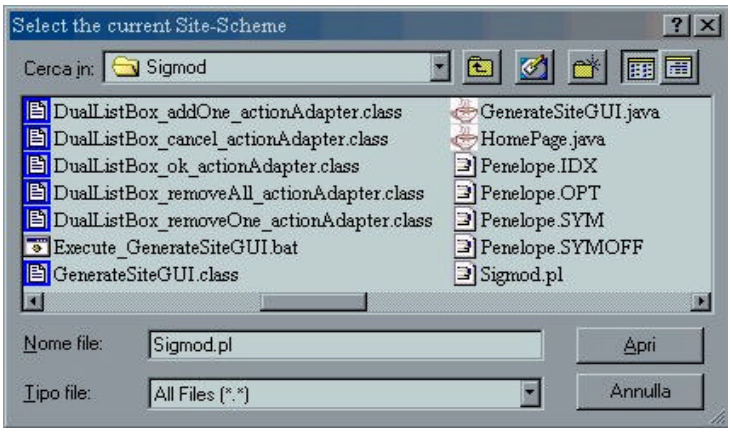


Figura 5.14: Finestra di dialogo.

Il terzo menu (Figura 5.15) permette di selezionare il formato HTML o XML e permette di generare sorgenti Java o pagine dinamiche per l'intero PL (Generate Program) oppure per un singolo page-scheme (Generate PageScheme)

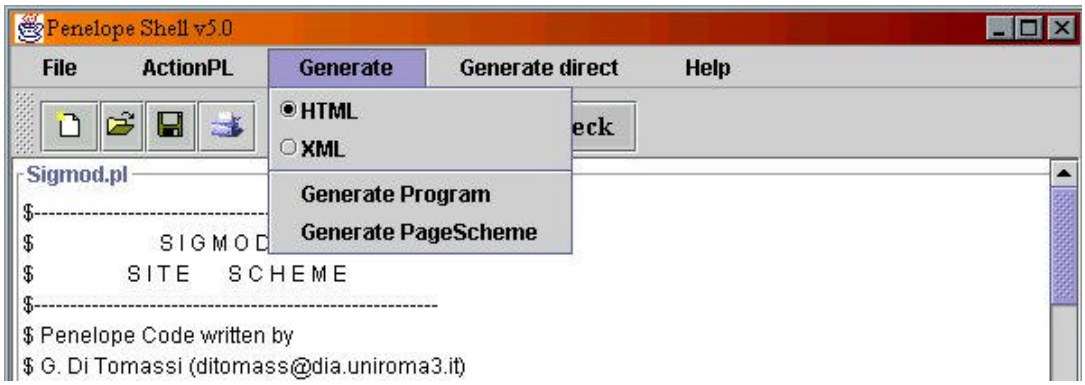


Figura 5.15: Menu "*Generate*" della PenelopeGUI

Il quarto menu (figura 5.16) consente di generare direttamente il sito in formato HTML, XML oppure in ASP.

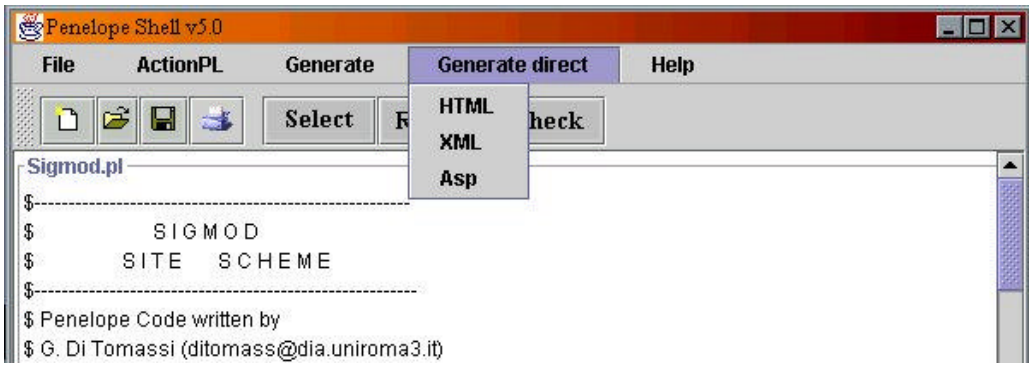


Figura 5.16: Menu "*Generate direct*" della PenelopeGUI

5.5 La GenerateSite GUI

Abbiamo visto che il nuovo prototipo Penelope è in grado di generare, sia programmi Java che una volta eseguiti e lanciati generano l'intero sito web, sia siti misti cioè programmi Java (quindi pagine statiche) e pagine JSP (quindi pagine dinamiche).

Quando si definisce nel PL un nuovo page-scheme se questo non viene dichiarato *DYNAMIC* allora viene generato un unico file Java che deve essere compilato al fine di produrre bytecode quindi eseguito per generare l'insieme di pagine web corrispondenti alla struttura definita. Per aiutare l'utente viene generata automaticamente anche una interfaccia grafica di sostegno, la *GenerateSiteGUI.*

Questa GUI è costituita da due listbox, la prima indicata con "Source list" contiene l'elenco dei page-scheme statici per i quali si è generato un programma java, la seconda indicata con "Destination list" conterrà i page-scheme per i quali si vogliono generare le pagine web.

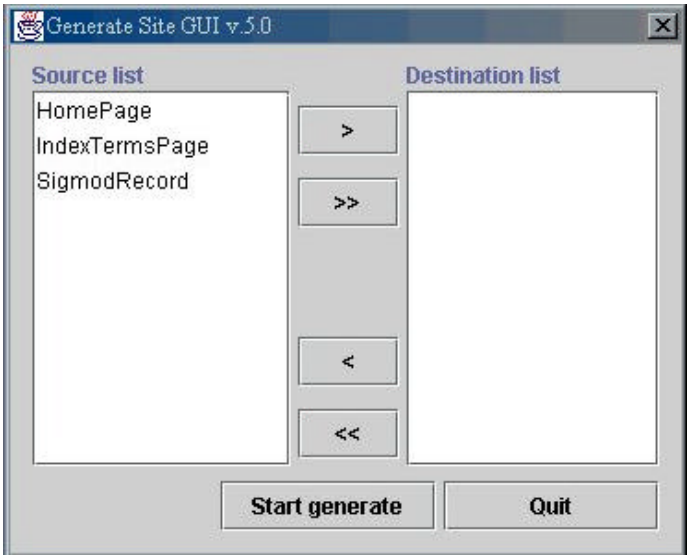


Figura 5.17: La GenerateSiteGUI

L'utente non deve far altro che selezionare un page-scheme dalla Source list e metterlo nella Destination list attraverso il bottone ">" allo stesso modo si può deselezionare un page-scheme attraverso il bottone "<". E' possibile anche selezionare ">>" e deselezionare "<<" tutti i page-scheme presenti nella Source list.

Una volta selezionato uno o più page-scheme premendo il bottone "Start generate" vengono compilati e poi eseguiti in sequenza i file selezionati.

La GenerateSiteGUI della figura 5.17 è stata generata automaticamente da Penelope in base al seguente PL:

```
$-------------------------------------------------------
$                  S I G M O D
$             S I T E     S C H E M E
$-------------------------------------------------------

SCHEME file:/
ON sigmodDiTomas

DEFINE PAGE Static HomePage : "Issues"
  ....
END

DEFINE PAGE Dynamic OrdinaryIssuePage : "Issues"
  ....
END

DEFINE PAGE Dynamic ProceedingsPage : "Issues"
  ....
END

DEFINE PAGE IndexTermsPage : "Issues"
  ....
END

DEFINE PAGE SigmodRecord : "Issues"
  ....
END
```

Come si può osservare, evidenziati in grassetto, sono stati definiti tre page-scheme statici HomePage, IndexTermsPage e SigmodRecord.