Progettazione di siti web centrati sui dati (Data-Intesive Web Applications) Dispensa del corso di Basi di dati e Web 2005-2006

Alberto Belussi
Università degli Studi di Verona
Dipartimento di Informatica

Giugno 2006

Indice

1 Introduzione

L'argomento trattato in questa dispensa riguarda la progettazione di un *sito web centrato sui dati*. In particolare l'obiettivo è quello di presentare allo studente una metodologia per la progettazione di un sito web centrato sui dati e di illustrare le soluzioni tecnologiche per la realizzazione di un sito di questa categoria. Gli esempi e gli esercizi proposte si focalizzeranno in particolare sulla progettazione e realizzazione di un sito contenente le informazioni che descrivono le attività di un percorso formativo.

Con il termine sito web centrato sui dati (o data-intesive web site) si intende indicare un sito web nelle cui pagine si presenta un insieme di *dati strutturati* che descrivono le informazioni di interesse per un certo ambito applicativo. Un insieme di dati strutturati è caratterizzato dall'avere una struttura ben definita e stabile nel tempo; in altre parole, esso si presenta come un insieme di elementi di informazione tutti con le medesime caratteristiche. Ad esempio, l'elenco telefonico è un insieme di dati strutturati visto che è costituito da una lista di elementi tutti con la stessa struttura: nome, cognome, indirizzo e numero di telefono. Ogni insieme di dati strutturati ben si presta alla rappresentazione in una base di dati (in accordo con quanto lo studente ha già visto nella prima parte del corso di Basi di dati e Web).

Nelle situazioni più frequenti la progettazione di un sito web centrato sui dati parte da uno stato iniziale in cui è già presente una base di dati, che gestisce le informazioni di interesse dell'organizzazione per la quale si vuole allestire un sito web. In particolare, spesso accade che l'insieme di dati strutturati da pubblicare coincida in buona parte con quanto già memorizzato e gestito da tale base di dati.

Si noti che il sistema che gestisce la base di dati è diverso da quello che governa il sito web: vale a dire, si tratta di due sistemi software distinti. Quindi la realizzazione di un sito web centrato sui dati si occuperà anche di far comunicare questi due sistemi.

In generale, quando i dati da gestire presentano una struttura ben definita e stabile nel tempo, è conveniente adottare per la loro gestione un sistema dedicato alla rappresentazione e interrogazione di basi di dati. Tale sistema, detto anche Data Base Management Systems (DBMS), consente infatti di gestire grandi quantità di dati persistenti garantendo affidabilità, privatezza e accesso efficiente. Quindi, anche quando nello stato iniziale del progetto non è presente un sistema per la gestione di basi di dati, supporremo che, per la realizzazione di un sito web centrato sui dati, tale sistema venga allestito e i dati vengano rappresentati in una base di dati. Vale a dire, l'ipotesi che vogliamo fissare è che una base di dati sia sempre presente.

In sintesi, nella architettura di un sito web centrato sui dati abbiamo due *attori*: il sistema che gestisce le richieste di pagine web che provengono dai browser (server web o http server[1]) e il sistema che gestisce la base di dati contenente i dati da pubblicare (server dati o database server). Si veda lo schema mostrato in Figura 1.

Nella figura si può notare che il server web (http server) contiene uno specifico modulo, detto *Engine for dynamic web pages*, che gestisce la generazione di pagine dinamiche e interagisce con il database server. Lo strumento che verrà utilizzato nelle esercitazioni svolte nel corso di Laboratorio di Basi di dati e Web contiene esattamente un modulo di questo tipo.

Nella prima parte del corso di Basi di dati e Web sono stati introdotti i concetti fondamentali relativi alla progettazione di una base di dati. Nelle prime lezioni del corso di Laboratorio si illustreranno le nozioni fondamentali per la corretta stesura di un documento HTML (*Hyper Text Markup Language* `http://www.w3c.org`) per la generazione di una pagina web statica. In questa dispensa vedremo come integrare queste conoscenze e, in particolare, come far sì che i dati estratti da una base di dati, presente

[1]dal nome del protocollo che sta alla base del suo funzionamento: Hyper Text Transfer Protocol - HTTP `http://www.w3c.org/Protocols`

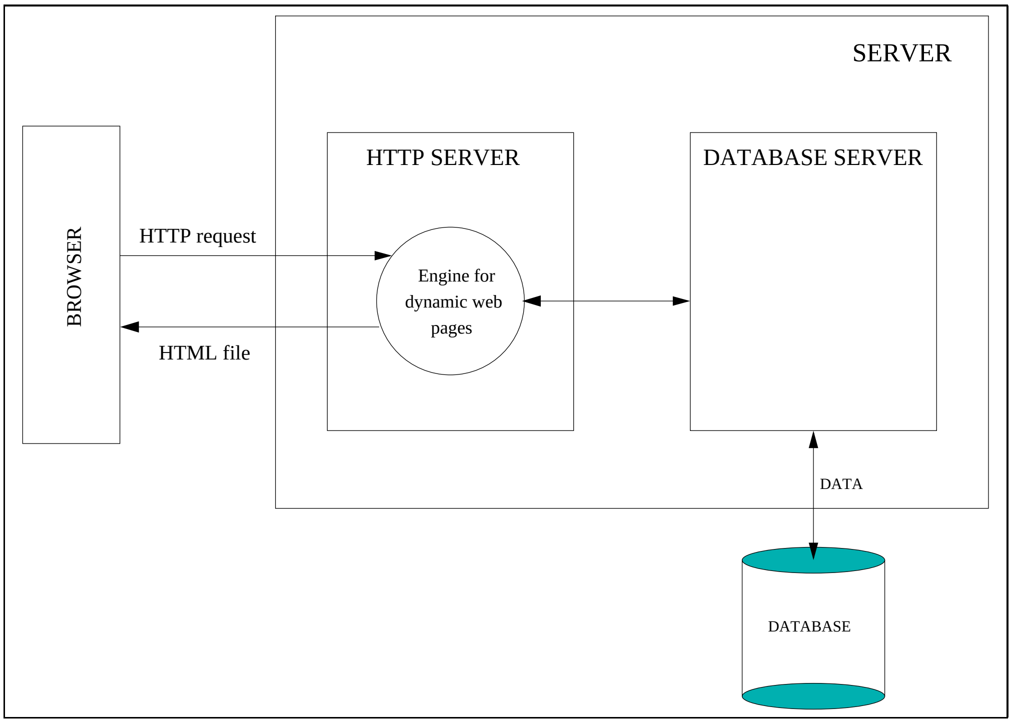

Figura 1: Architettura di un sistema che gestisce un sito web centrato sui dati.

su un server dati, possano essere automaticamente trasferiti in un file HTML gestito da un server web. Infatti, la caratteristica fondamentale dei siti centrati sui dati è la dinamicità del contenuto delle pagine che li compongono. Mentre in un sito tradizionale le pagine web, che lo costituiscono, sono file statici scritti nel linguaggio HTML e il server web, alla richiesta di una certa pagina da parte di un browser, non fa altro che spedire il file così come è memorizzato nel server web stesso, nei siti centrati sui dati la pagina web viene generata al momento della richiesta combinando lo schema base della pagina (parte statica) con i dati estratti dalla base di dati (parte dinamica) e producendo, solo al momento della richiesta, il file HTML che verrà inviato al browser.

In questa dispensa si vuole presentare una metodologia per la progettazione di pagine web dinamiche e si intende analizzare la tecnologia che consente di realizzarle.

Per affrontare lo studio di questa dispensa sono necessari i seguenti prerequisiti:

- una sufficiente conoscenza dei principali tag del linguaggio HTML (*Hyper Text Markup Language*);
- una buona conoscenza delle basi di dati e in particolare si richiede la capacità di: progettare lo schema concettuale di una base di dati nel modello Entità-Relazione (ER), tradurre lo schema ER in uno schema relazionale per la rappresentazione dei dati a livello logico e creare/interrogare una base di dati relazionale usando il linguaggio SQL.

Le successive sezioni della dispensa sono strutturate come segue: nella seconda, terza e quarta sezione si trattano le tecnologie per la realizzazione di un sito web centrato sui dati, nella quinta sezione si propone una metodologia per la progettazione si siti web centrati sui dati; infine, nell'ultimza sezione si presenta un'architettura software di riferimento basata sul paradigma MVC per siti web centrati sui dati.

2 Le tecnologie per la realizzazione di un sito web centrato sui dati

In questo capitolo si presenteranno brevemente le caratteristiche delle principali tecnologie oggi disponibili per realizzare un sito web centrato sui dati. In particolare, l'obiettivo è quello di presentare un insieme di concetti comuni a tutte le tecnologie per la generazione di pagine web dinamiche e di focalizzare poi sulla tecnologia *JavaServlet* di Sun. Infatti, come descritto nell'introduzione, una sito web centrato sui dati è costituito da un insieme di *pagine web a contenuto dinamico* (presentati dati estratti da una base di dati), dette più semplicemente *pagine web dinamiche*

Poiché non si ha la pretesa di coprire interamente l'argomento relativo alla generazione di pagine web dinamiche, che comprende necessariamente anche un corso di programmazione con linguaggi imperativi, in questo capitolo si presenteranno in modo informale e discorsivo molti concetti che in una trattazione più approfondita avrebbero richiesto molto più spazio e rigore.

Il capitolo è organizzato come segue: nella prima sezione si descrive brevemente il protocollo HTTP, che regola l'interazione tra browser e web server presentando, in particolare, come si realizza tale interazione nel caso in cui il sito sia costituito da pagine web dinamiche. Nella sezione successiva si presentano le principali differenze tra le tecnologie più diffuse per la realizzazione di pagine web dinamiche.

2.1 Interazione tra browser e web server

La navigazione sulla rete Internet si basa sull'interazione tra l'applicazione di visualizzazione di ipertesti (detta comunemente *browser*) e i web server connessi alla rete Internet. Ogni volta che sul nostro browser digitiamo un indirizzo di una pagina web (ad esempio, `http://www.scienze.univr.it`) stiamo instaurando con uno specifico web server una interazione che prevede lo scambio di dati sulla rete.

Il protocollo che regola tale interazione si chiama Hyper Text Transfer Protocol (HTTP). Tale protocollo, come indica il suo nome, regola lo scambio di ipertesti tra browser e web server. Non si vuole qui analizzare nel dettaglio tale protocollo, ma soltanto chiarire le caratteristiche fondamentali del suo funzionamento e, in particolare, descrivere cosa accade quanto si è in presenza di pagine web dinamiche.

Va innanzitutto precisato che l'interazione tra browser e web server si ha, non solo quando si digita esplicitamente l'indirizzo di una pagina web (tale indirizzo viene anche detto Universal Resource Locator - URL), ma anche quando si clicca su un link presente nella pagina visualizzata correntemente dal browser. Infatti, se ricordiamo la sintassi per la specifica di un link nel linguaggio HTML, essa contiene sempre la specifica di un URL. Si noti che un URL può rappresentare in generale richieste basate su protocolli diversi da HTTP (ad esempio basate su protocollo FTP - File Transfer Protocol). L'indicazione del protocollo occupa infatti la prima parte dell'URL la quale, per le pagine web, inizia sempre con la stringa http://, visto che il protocollo usato in questo caso è esattamente l'HTTP.

Quando il sito è costituito da pagine statiche, l'interazione browser/web server risulta molto semplice. I passi in cui si articola tale interazione sono i seguenti:

- Il browser invia la richiesta di una pagina web al server specificando, oltre all'URL, una serie di altre informazioni tra cui il tipo di richiesta (fra i tipi possibili segnaliamo il tipo di richiesta GET e il tipo di richiesta POST).
- Il web server riceve la richiesta del browser e individua il file HTML che corrisponde alla richiesta.

- Il web server invia il file HTML individuato al browser (in molti casi la presenza nella pagina web di immagini o di una struttura a frame richiede in realtaù più di un'interazione di questo tipo per la visualizzazione dell'intera pagina sul browser).

Per la gestione di pagine web statiche il server non necessita di particolari strumenti, ma è sufficiente una struttura di cartelle (dette anche folder o directory), dove memorizzare tutti i file HTML che costituiscono le pagine web del sito (vedi Figura 2).

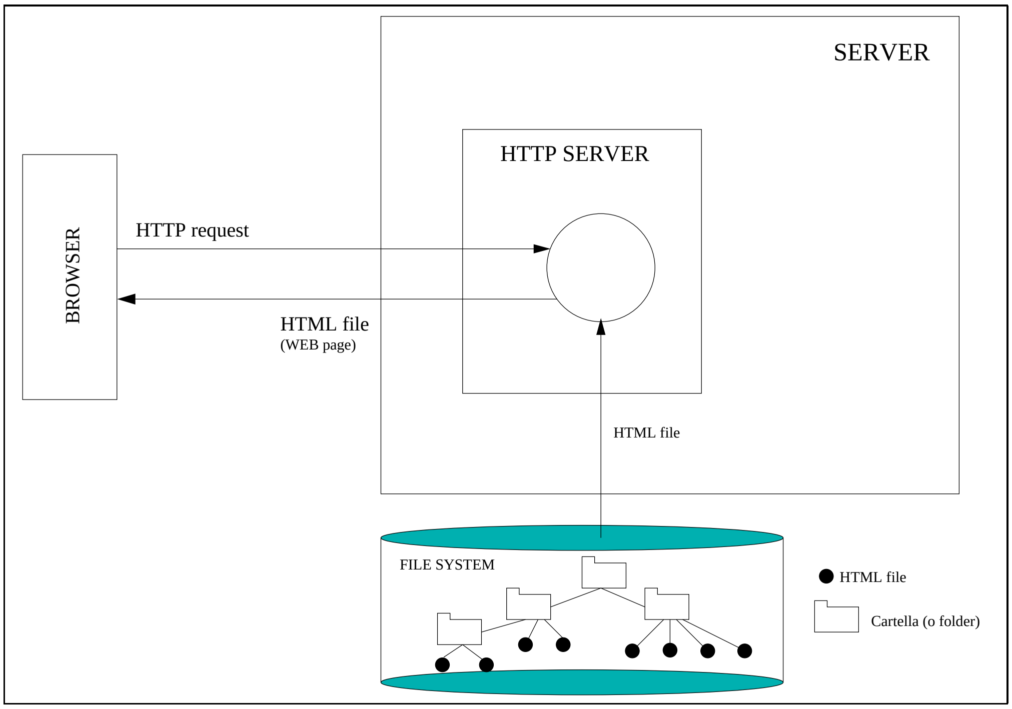

Figura 2: Interazione browser/web server per lo scambio di una pagina web statica.

Cosa accade invece quando il sito è costituito da pagine web dinamiche?

L'interazione in questo caso è fondamentalmente simile a quella che si ha per le pagine statiche con una importante differenza: oltre a comunicare l'URL dello schema di pagina richiesto, il browser invia al web server anche alcuni parametri per individuare i dati da inserire nello schema di pagina e produrre la pagina web finale. Con *schema di pagina* si intende qui indicare la parte statica della pagina web da generare. Infatti, la pagina web dinamica viene costruita al momento della richiesta (on the fly) combinando uno schema statico HTML con il risultato di un'estrazione di informazioni da una base di dati gestita da un database server.

In questo caso i passi in cui si articola l'interazione browser/web server e l'evasione della richiesta da parte del server sono i seguenti:

- Il browser invia la richiesta di una pagina web al server specificando un URL contenente uno o più parametri. L'invio dei parametri può avvenire in due modi: a) in una richiesta di tipo GET i parametri sono accodati all'URL secondo la sintassi seguente:

  `http://www. ..... /<nome_schema_pagina>?`

```
<nome_parametro_1>=<valore_parametro_1>&....
&<nome_parametro_n>=<valore_parametro_n>
```

ad esempio:

`http://www.sitoaziendale.com/Dipendente?cod_dip=3`

b) in una richiesta di tipo POST i parametri sono inviati al web server in modo indipendente dall'URL e non sono quindi visibili nella stringa che rappresenta l'URL stessa.

- Il web server riceve la richiesta del browser, individua lo schema di pagina richiesto e costruisce la/le interrogazione/i da inviare al database server in base ai valori dei parametri ricevuti.
- Il database server interagisce con la base di dati per rispondere alle interrogazioni ricevute dal web server e invia a quest'ultimo il risultato.
- Il web server riceve il risultato delle interrogazioni dal database server e elabora tale risultato per inserirlo nello schema di pagina in modo da produrre la pagina web finale.
- Il web server invia al browser il file HTML che rappresenta la pagina web prodotta al passo precedente. Si noti che, il browser non è in grado di riconoscere, dal codice HTML ricevuto, se la pagina web è statica o dinamica.

In Figura 3 si mostra una esempio di interazione browser/web server in presenza di pagine web dinamiche.

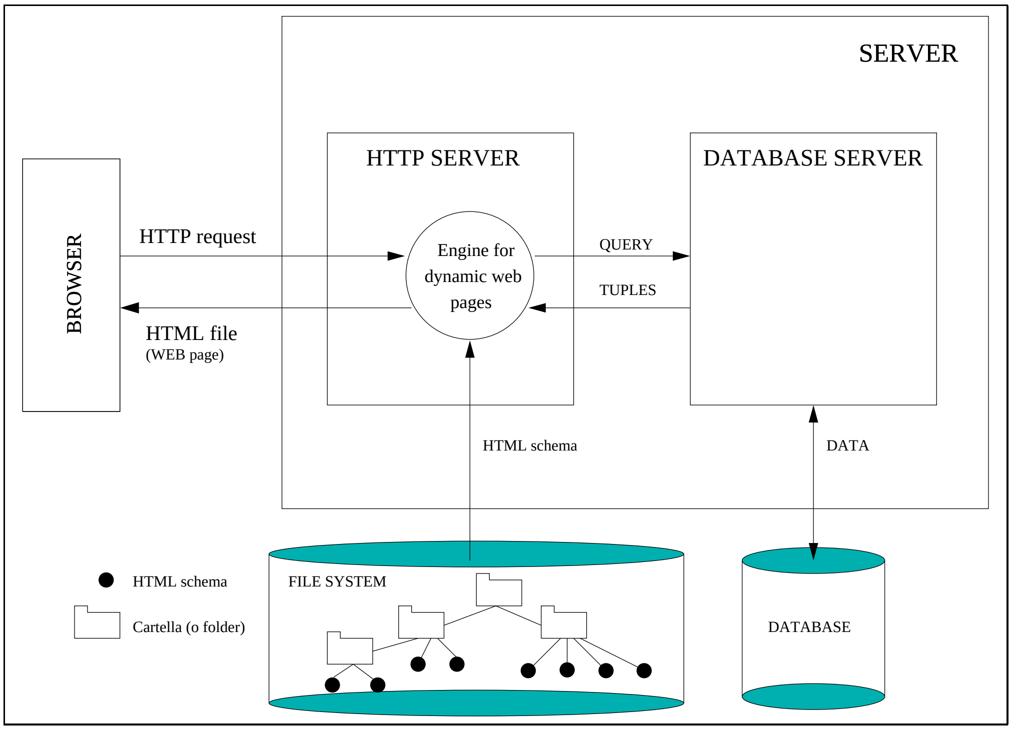

Figura 3: Interazione browser/web server per lo scambio di una pagina web dinamica.

Nella prossima sezione vediamo quali sono le tecnologie più diffuse che vengono utilizzate per la realizzazione pagine dinamiche.

2.2 Le tecnologie per la realizzazione di pagine dinamiche

In questa sezione si presentano molto brevemente le tecnologie attualmente disponibili per generare pagine web dinamiche.

Le soluzioni disponibili si suddividono storicamente in due categorie:

- **Common Gateway Interface (CGI)**: questa soluzione è stata la prima ad essere adottata sui web server per la generazione di pagine web dinamiche. Essa consente al web server di attivare altri programmi sul server e di comunicare con tali programmi inviando una richiesta di elaborazione e ottenendo il risultato della stessa elaborazione dal programma. Per ogni richiesta proveniente dal browser nell'approccio CGI viene attivato un nuovo programma (processo) che si occupa di ricevere l'interrogazione, collegarsi al database server, eseguire l'interrogazione e restituire il risultato al server web. L'attivazione di un nuovo processo a fronte di ogni nuova richiesta è il principale svantaggio di questa soluzione; infatti, tale attivazione richiede tempo e risorse di memoria e, su un web server con traffico elevato, ciò potrebbe portare velocemente al blocco del sistema. La soluzione CGI, inoltre, presenta alcuni problemi di sicurezza, in quanto risulta maggiormente vulnerabile esattamente nella fase di attivazione di processi sul sistema operativo. Infine, la stesura di pagine web dinamiche richiede necessariamente in questo approccio la conoscenza approfonditia di un linguaggio di programmazione imperativo (ad esempio, C o C++ o Java o altro).
- Approcci basati sull'estensione del web server: queste soluzioni prevedono di estendere il web server con moduli software aggiuntivi che siano in grado di gestire l'esecuzione di codice senza attivare nuovi processi e senza interagire direttamente con il sistema operativo del server. Descriviamo brevemente alcuni di questi approcci.
  - **JavaServlet** di Sun: questa soluzione prevede di estendere il web server con un interprete Java (Java Virtual Machine - JVM) per l'esecuzione di programmi Java sul server, di qui il nome *Servlet* coniato a partire da *Applet.* Con questo approccio quindi il web server è in grado di eseguire porzioni di codice in Java senza interagire con il sistema operativo per attivare nuovi processi.

    L'esecuzione di una servlet per l'evasione di una richiesta HTTP giunta dal browser richiede l'attivazione di un nuovo thread nell'unico processo che gestisce la Java Virtual Machine: un thread, a differenza di un processo, non richiede uno spazio di memoria dedicato e la sua attivazione risulta quindi molto meno dispendiosa. I vantaggi delle JavaServlet rispetto all'approccio CGI risiedono quindi nei seguenti punti: a) richiedendo solo l'attivazione di thread, e non di nuovi processi, l'esecuzione di Servlet risulta meno pesante per il sistema operativo e più scalabile rispetto a CGI; b) adottando il linguaggio Java le Servlet godono dei vantaggi di tale piattaforma e in particolare: portabilità e ricchezza di librerie riusabili.

    L'unico svantaggio che rimane sia nelle Servlet che nell'approccio CGI riguarda la stesura dei programmi, in quanto la generazione di codice HTML, che è il naturale risultato dei programmi scritti in tale contesto, avviene attraverso istruzioni esplicite di stampa di stringhe, sia per la parte statica della pagina, sia per la parte dinamica (ad esempio nelle Servlet spesso si troveranno istruzioni del tipo: `out.println(<p>One line of HTML</p>);`. Tale svantaggio viene superato dalla successiva categoria di tecnologie.
  - **Template systems**: sotto questo nome troviamo tutte le nuove tecnologie per produrre pagine web dinamiche che si basano su un nuovo approccio. Si tratta di:
    * Microsoft Active Server Pages (ASP) (`http://msdn.microsoft.com/`),
    * open-source software Hytertext PreProcessor (PHP) (`http://www.php.net/`),
    * Java Server Pages (JSP) (`http://java.sun.com/products/jsp/`) e altri.

L'approccio comune di tutte queste tecnologie è quello di realizzare pagine web dinamiche semplicemente introducendo nei file HTML del codice (pezzi di programma in un linguaggio imperativo), che consente di rendere dinamico il contenuto della pagina. Ad esempio, se si deve inserire l'elenco degl studenti ad un certo punto di un file HTML, si inserirà in quel punto un elemento (o tag) particolare che conterrà il codice in grado di eseguire l'interrogazione che estrae gli studenti. In questo caso il file HTML cambia tipologia in quanto non contiene più solo tag HTML, ma anche questi elementi (o tag) speciali che gestiscono le parti dinamiche.

Ad esempio, se si utilizza Microsoft Active Server Pages il file diventa un file ASP oppure se si usa Java Server Pages diventa un file JSP. Non deve essere fatto niente di più. Sarà il web server che, alla richiesta di un certo schema di pagina (file ASP o JSP), eseguirà il codice e sostituirà nel file, il codice stesso, con il risultato dell'esecuzione, producendo un puro file HTML.

Ad esempio, la pagina web dinamica che pubblica l'elenco degli studenti può essere generata dal seguente file JSP, che contiene sia lo schema di pagina sia il codice che consente la generazione delle parti dinamiche. Il codice è scritto nel linguaggio JAVA ed è contenuto nei tag JSP che sono sempre delimitati dai simboli seguenti: `<% ... %>`. Si presenta tale codice per dare allo studente un idea di come si applica l'approccio dei Template systems. Non si intende descrivere nel dettaglio il codice JAVA contenuto in tale esempio. Per approfondimenti si vedano i seguenti testi [FKB02, H01].

```
<%@ page import="studenteBean" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
  <title>
    Elenco Studenti
  </title>
</head>

<body>
  <h1>Lista studenti</h1>

  <% Vector lista = (Vector)request.getAttribute("listaStudenti");
     StudenteBean studente = new StudenteBean(); %>

  <table border="1">
  <tr>
  <td><strong>Matricola</strong> </td>
  <td><strong>Cognome</strong> </td>
  <td><strong>Nome</strong></td>
  <td><strong>Facolt&agrave;</strong></td>
  <td><strong>Data di nascita</strong></td>
  <td><strong>Crediti ottenuti</strong></td>
  </tr>

  <tr>
  <% for (int i=0; i<lista.size(); i++) {
   studente = (StudenteBean) lista.get(i); %>
```

```
  <td><%= studente.getMatricola() %></td>
  <td><%= studente.getCognome() %></td>
  <td><%= studente.getNome() %></td>
  <td><%= studente.getFacolta() %></td>
  <td><%= df.format(studente.getDataNascita()) %></td>
  <td><%= studente.getCreditiOttenuti() %></td>

  </tr>
  <% } %>
  </table>

  <p>
  <a href="http://validator.w3.org/check/referer">Valida il documento</a>
  </p>
</body>
</html>
```

Va precisato che l'adozione di un Template system su un web server richiede sempre l'installazione di uno specifico software per la gestione delle nuove tipologie di file (ASP o JSP) che il sistema deve gestire. Tale software aggiuntivo solitamente chiamato *Engine* (JSP engine o ASP engine) è esattamente quel modulo che abbiamo chiamato in generale *engine for dynamic web pages* nell'introduzione della dispensa.

Come si nota, l'uso di qualsiasi tecnologia della categoria Template systems richiede la conoscenza di un linguaggio di programmazione imperativo e, quindi, non può essere facilmente utilizzata da chi non ha già una buona esperienza di programmazione. In realtà alcuni sistemi propongono ambienti per lo sviluppo di pagine web dinamiche, dove la conoscenza dei concetti della programmazione imperativa è ridotta al minimo. È il caso ad esempio del sistema Zope. In questi sistemi la specifica delle parti dinamiche non viene completamente delegata al linguaggio di programmazione e racchiusa in un solo tipo di tag speciale, ma vengono invece forniti diversi tipi di tag speciali, che consentono di specificate i più comuni tipi di elaborazioni necessarie per la generazione delle parti dinamiche senza richiedere la stesura di programmi.

Vediamo ora come realizzare le parti dinamiche di una pagina web usando la tecnologia delle *JavaServlet* di Sun. Si illustra, in particolare, un uso elementare delle servlet nella realizzazione di un insieme ridotto di pagine dinamiche; applicazioni web più complesse e articolate richiedono l'applicazione di architetture software più avanzate, quali quelle suggerite dal paradigma MVC (Model View Control) [ACM01].

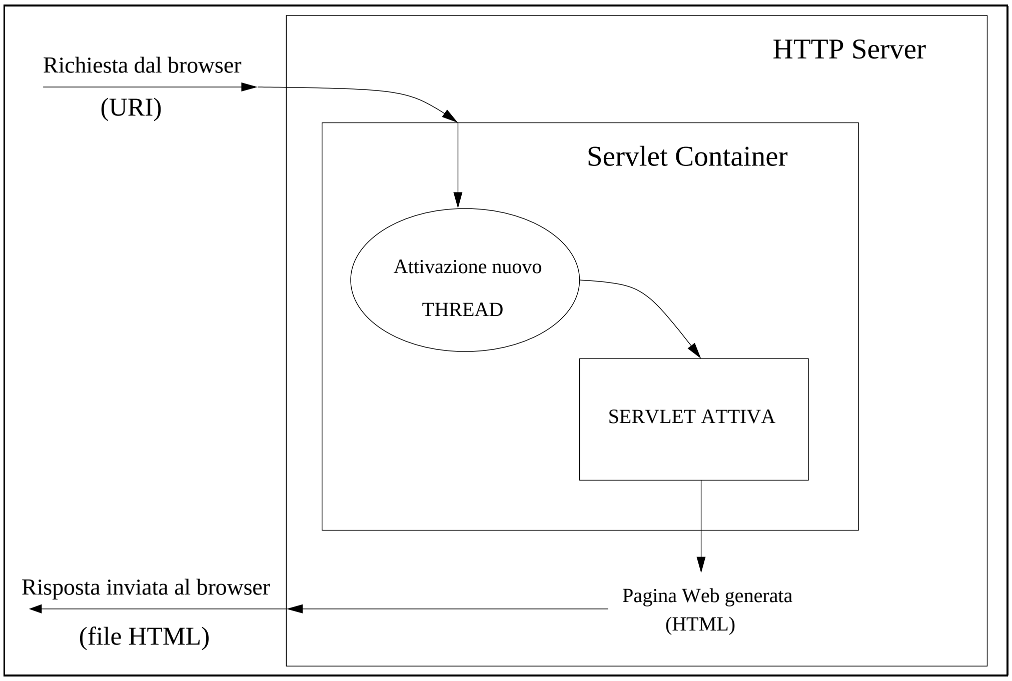

Figura 4: Architettura di base di un JavaServlet engine.

3 La realizzazione di una pagina web dinamica usando la tecnologia JavaServlet di Sun

Prima di trattare nel dettaglio la realizzazione di pagine dinamiche con la tecnologia JavaServlet di Sun, presentiamo brevemente le caratteristiche di un *JavaServlet engine*, vale a dire, del modulo software che viene attivato sul web server per consentire l'esecuzione di servlet. Solitamente l'installazione di tale modulo ha come effetto la generazione sul server di un albero di directory dove lo sviluppatore dell'applicazione andrà a collocare: i file contenenti le servlet, le classi Java di supporto alle servlet, i file HTML statici, ecc..

L'attivazione di un *JavaServlet engine* sul web server corrisponde all'attivazione di un *Servlet Container*, che implementa una Java Virtual Machine (JVM), vale a dire, un interprete JAVA. Il comportamento di un *Servlet Container* è mostrato in Figura 4. Si noti che l'arrivo di una richiesta HTTP al web server, il *Servlet Container*, che implementa una JVM, attiva un thread che esegue la servlet richiesta e produce il codice HTML che viene restituito al browser chiamante. Il thread a differenza del processo non richiede uno spazio di memoria dedicato, ma condivide memoria con altri thread, ciò consente di ridurre notevolmente il tempo di attivazione rispetto al processo.

Vediamo ora come si scrive una servlet. Una servlet è una classe Java ottenuta estendendo la classe `HttpServlet`. Tale classe contiene alcuni metodi predefiniti per trattare i diversi tipi di richiesta HTTP che possono giungere dal browser. Esistono ad esempio i metodi `doGet` e `doPost` per rispondere alle richieste GET e POST (la maggior parte delle richieste che vengono gestite da servlet sono di tipo GET o POST). Di seguito si mostra una servlet che produce un file HTML contenente la classica stringa Hello World.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
       public void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                   throws IOException, ServletException {

                   response.setContentType("text/html");
                   PrintWriter out = response.getWriter();
                   String docType =
                          "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0"+
                          " transitional\//EN\">\n";
                   out.println(docType +
                               "<html>\n" +
                               "<head><title>Hello WWW</title></head>\n" +
                               "<body>\n" +
                               "<h1>Hello World!<h1>\n" +
                               "</body></html>");
                   }
}
```

Si noti che la servlet `HelloWorld` viene definita come estensione della classe `HttpServlet`. Nella servlete viene ridefinito il metodo `doGet`: tale metodo verrà invocato nel caso in cui giunga al web server una richiesta HTTP di tipo GET indirizzata alla servlet `HelloWorld`. Infine, il metodo `doGet` produce codice HTML che viene preparato attraverso l'istruzione `out.println(...codice HTML...)` e rispedito al browser dal web server.

Nella sottosezione successiva si descrivono in maggior dettaglio gli oggetti `request` e `response` che compaiono come parametri del metodo `doGet`. Gli stessi parametri compaiono anche nel metodo `doPost` che gestisce invece le richieste di tipo POST.

3.1 Gli oggetti delle classi `HttpServletRequest` e `HttpServletResponse`

I metodi `doGet` e `doPost` della classe `HttpServlet` hanno due parametri predefiniti: l'oggetto `request` e l'oggetto `response`. Tali oggetti presentano alcune proprietà ed alcuni metodi che sono di estrema utilità nella scrittura di una servlet. Mostriamo di seguito per ognuno dei due oggetti alcuni dei metodi più utili.

- `HttpServletRequest request`: questo oggetto consente di accedere alle informazioni che riguardano la richiesta HTTP giunta al web server indipendentemente dal tipo di richesta (GET o POST). In particolare, `request` mette a disposizione, tra gli altri, i seguenti metodi:
  - `request.getParameter(NomeParametro)`: restituisce una stringa (oggetto Java di tipo `String`) che rappresenta il valore della prima occorrenza del paramentro `NomeParametro` presente nella richiesta. Se il parametro non esiste restituisce `NULL`, infine, se il parametro esiste ma non ha valore, restituisce la stringa nulla.

  - `request.getParameterValues(NomeParametro)`: restituisce un array di stringhe che rappresentano tutti i valori del parametro `NomeParametro` presenti nella richiesta. Restituisce un array con un elemento stringa vuota, se il parametro esiste ma non gli sono assegnati valori.
  - `request.getParameterNames()`: restituisce un enumerazione (`Enumeration`) di stringhe contenente i nomi dei parametri presenti nella richiesta.

- `HttpServletResponse response`: questo oggetto consente di preparare la risposta da inviare al browser sotto forma di codice HTML. In particolare, `response` mette a disposizione, tra gli altri, i seguenti metodi:
  - `response.getWriter()`:
    restituisce un oggetto della classe `PrintWriter` al quale è possibile inviare stringhe di caratteri con il metodo `println(StringaCaratteri)`
  - `response.getBufferSize()`: restituisce la dimensione del buffer gestito dall'oggetto `PrintWriter`. Tale buffer rappresenta l'area di memoria dove viene preparato il codice HTML da spedire al browser: il buffer viene riempito attraverso l'istruzione `out.println(StringaCaratteri)` dove `out` è un oggetto della classe `PrintWriter`. Quanto il buffer è pieno il suo contenuto viene spedito al browser.
  - `response.setBufferSize()`:
    consente di fissare la dimensione del buffer usato dall'istruzione `out.println(StringaCaratteri)`. Tale metodo va invocato prima di chiamare `out.println(StringaCaratteri)`.

Esistono molti altri metodi negli oggetti `request` e `response`, la cui descrizione può essere rintracciata sul sito della Sun (`http://java.sun.com/products/servlet`), tuttavia, come mostrato nell'esempio di servlet precedente, bastano quelli qui elencati per produrre una servlet funzionante.

Ovviamente il corpo dei metodi `doGet` o `doPost` può essere un programma Java molto complesso ed eseguire elaborazioni diverse. In questa dispensa siamo interessati alla realizzazione di pagine web dinamiche, quindi il codice Java che va inserito nei metodi suddetti deve interagire con un database server, eseguire interrogazioni in base ai parametri ricevuti e preparare in codice HTML che presenta il risultato dell'interrogazione.

Come avviene l'interazione tra la l'ambiente Java e un database server? Anche questo problema viene risolto da un insieme di classi Java già predisposte per gestire tale interazione. Tale insieme di classi costituisce la libreria JDBC (Java Database Connectivity) oggetto della prossima sezione.

3.2 JDBC: una libreria di classi Java per l'interazione con un database server

La libreria JDBC (Java Database Connectivity) contiene un insieme di classi Java che mette a disposizione dello sviluppatore di applicazioni Java una serie di strumenti per l'interazione con un qualsiasi database server, purché questo fornisca un *driver JDBC*.

Un *driver JDBC* è un modulo software, dedicato ad uno specifico database server D, in grado di tradurre tutte le fuzionalità fornite da JDBC in comandi del linguaggio di interrogazione adottato da D (nella maggior parte dei casi il linguaggio è SQL).

Senza presentare tutti i dettagli delle classi di JDBC, che si possono trovare sul sito della Sun (`http://java.sun.com/products/jdbc`), mostriamo di seguito i sette passi necessari per interrogare all'interno di un metodo Java (e quindi anche nei metodi di una servlet) un database server e processare il risultato dell'interrogazione.

1. **Attivazione del driver JDBC**: prima di attivare una connessione con il database server è necessario caricare il driver JDBC per il database server stesso. A tale scopo basta caricare la classe Java corrispondente con la seguente istruzione:

   ```
   import java.sql.*

   Class.forName(NomeDriverJDBC)
   ```

   dove `NomeDriverJDBC` per il database server utilizzato nelle esercitazioni di laboratorio è uguale alla stringa: `org.postgresql.Driver`.

2. **Preparazione di una connessione con un database server**: prima di attivare la connessione è opportuno preparare in alcune variabili i parametri della connessione. In particolare, per attivare una connessione occorre specificare: l'URL del database server, il nome della base di dati a cui ci si vuole connettere, l'utente e la password da usare nella connessione. Per le esercitazioni di laboratorio vanno quindi definite le seguenti variabili:

   ```
   String URL = "jdbc:postgresql://DatabaseServer/BaseDiDati";
   String user = "UtentePostgresql";
   String passwd = "PasswordUtentePostgresql";
   ```

   dove `DatabaseServer` è il nome del database server e `BaseDiDati` è il nome della base di dati (ad esempio, `esercitazioni`).

3. **Apertura della connessione**: l'apertura di una connessione si esegue creando un oggetto della classe `Connection` con la seguente istruzione (si ipotizza di aver impostato correttamente le variabili `URL`, `user` e `passwd` come mostrato al punto precedente):

   ```
   try {
       Connection con = DriverManager.getConnection(URL, user, passwd);
       } catch (SQLException sqle) {
         System.err.println("DriverManager non trovato: "+sqle.getMessage());
   }
   ```

   Si noti che l'attivazione della connessione avviene attraverso il metodo statico `getConnection` della classe `DriverManager`; tale classe della libreria `java.sql` fornisce i metodi per la gestione dei driver JDBC attivi sulla JVM. Si noti inoltre che la chiamata al metodo `getConnection` viene eseguita all'interno di un `try ... catch` per catturare e gestire correttamente l'eventuale fallimento della connessione. Se non si è verificato un errore dopo l'esecuzione del metodo `getConnection`, allora è stata attivata una connessione con il database server.

4. **Preparazione di un'interrogazione da sottomettere al database server**: per sottomettere un'interrogazione al database server sul quale è stata aperta una connessione, è necessario creare un oggetto della classe `Statement` e preparare una stringa con l'espressione SQL che rappresenta l'interrogazione. L'esempio seguente mostra le istruzioni necessarie per la preparazione di un'interrogazione sulla tabella d'esempio `STUDENTE(Matricola, Cognome, Nome, CreditiOttenuti)`:

   ```
   Statement stat = con.createStatement();
   String queryExp  = "SELECT Matricola, Cognome, Nome FROM STUDENTE";
   String updateExp = "UPDATE STUDENTI SET CreditiOttenuti = CreditiOttenuti + 4 "+
                      "WHERE Matricola = 'IN000333'";
   ```

   A questo punto l'interrogazione può essere inviata al database server.

5. **Invio dell'interrogazione al database server**: per inviare l'interrogazione attraverso l'oggetto `Statement stat` creato al punto precedente al database server su cui è stata aperta una connessione occorre invocare i metodi di tale oggetto, in particolare: per eseguire un'interrogazione è possibile invocare il metodo `executeQuery`, ad esempio:

   ```
   ResultSet res = stat.executeQuery(queryExp);
   ```

   invece, per eseguire un comando di aggiornamento (`INSERT`, `UPDATE` o `DELETE`), va utilizzato il metodo `executeUpdate`, ad esempio:

   ```
   stat.executeUpdate(updateExp);
   ```

   Va sottolineato che il metodo `executeQuery` restituisce un oggetto di tipo `ResultSet` nel quale è contenuto il risultato dell'interrogazione, mentre il metodo `executeUpdate` restituisce un intero che indica il numero di tuple aggiornate, inserite o cancellate, a seconda del comando SQL eseguito.

6. **Elaborazione del risultato dell'interrogazione**: questa fase riguarda esclusivamente il caso in cui si sia eseguita un'interrogazione e vada elaborato il risultato ottenuto; ciò accade, ad esempio, quando sia necessario generare il codice HTML che presenta i dati estratti. Per accedere al risultato dell'interrogazione è necessario invocare i metodi dell'oggetto `ResultSet` restituito. Tale oggetto, che chiamiamo `res`, rappresenta il risultato dell'interrogazione come insieme di righe o tuple, dove una di tali righe, all'inizio la prima, risulta essere la riga corrente o riga puntata dal cursore: su tale riga agiscono i metodi invocati su `res`. Per l'elenco completo dei metodi si veda al solito il sito della Sun (`http://java.sun.com/products/jdbc`), tra gli altri segnaliamo il seguenti:

   - `res.next()`: consente di spostare il cursore sulla riga successiva alla riga corrente.
   - `res.getXxxx(Attribute)` (`Xxxx` sta per un tipo base Java, vale a dire: `String`, `Int`, ecc..): consente di estrarre il valore dell'attributo `Attribute`, dove `Attribute` può essere o la stringa che rappresenta il nome dell'attributo (tale nome viene definito nella clausola `SELECT` dell'interrogazione SQL eseguita) o l'indice che indica la posizione dell'attributo nella clausola `SELECT` dell'interrogazione SQL eseguita.
   - `res.findColumn(Attribute)`: restituisce un intero che rappresenta la posizione dell'attributo di nome `Attribute` nella clausola `SELECT` dell'interrogazione SQL eseguita.
   - `res.wasNull()`: restituisce un valore booleano che indica se l'ultima esecuzione di un metodo `res.getXxxx(Attribute)` ha restituito un valore SQL nullo.

   Ad esempio, considerando l'esecuzione dell'interrogazione mostrata al punto precedente:

   ```
   ResultSet res = stat.executeQuery(queryExp);
   ```

   il risultato potrebbe essere elaborato come segue, l'intenzione è quella di generare il codice HTML per mostrare l'elenco degli studenti nella pagina web.

   ```
   while(res.next()) {
         out.println(res.getString(1) + " "
                   + res.getString("Cognome") + " "
                   + res.getString("Nome"));
   }
   ```

7. **Chiusura della connessione**: al termine della servlet è possibile chiudere la connessione con il database server invocando un metodo sull'oggetto `con` della classe `Connection` generato precedentemente.

```
con.close();
```

Si noti che tale chiusura può essere evitata se esiste nell'applicazione una gestione centralizzata delle connessioni, realizzata ad esempio con uno strumento Java che gestisce pool di connessioni a database server.

Per facilitare la specifica di interrogazioni parametriche JDBC mette a disposizione una strada alternativa rispetto a quella presentata al punto 5. Invece di invocare il metodo `createStatement()` sull'oggetto `con` e manipolare poi l'oggetto `stat` di tipo `Statement`, è possibile generare un oggetto della classe `PreparedStatement` attraverso il metodo `prepareStatement(SQLquery)` del medesimo oggetto `con`. Tale metodo consente di sottomettere al database server la stringa `SQLquery` dove alcune costanti possono mancare. Le costanti mancanti sono rappresentate nella stringa `SQLquery` dal carattere `?`. Successivamente con i metodi della classe `PreparedStatement` è possibile sostituire i simboli `?` con i valori corretti. Tali metodi hanno la seguente forma sintattica:

```
PreparedStatement pstm = con.prepareStatement(query);
pstm.setXxxx(i, valore);
```

dove `Xxxx` sta per un tipo base Java, vale a dire: `String`, `Int`, ecc.., `i` indica quale simbolo `?` si sta sostituendo (1: il primo, 2: il secondo, ecc..), infine, `valore` rappresenta il valore inserire al posto del simbolo `?`. Si veda il seguente esempio:

```
sql  = " SELECT Cognome, Nome ";
sql += " FROM STUDENTE ";
sql += " WHERE Matricola = ? ";

pstmt = con.prepareStatement(sql);
pstmt.setString(1, "IN000001");
rs=pstmt.executeQuery();
```

Si noti che con l'uso di oggetti della classe `PreparedStatement` si ottiene un ulteriore vantaggio: la specifica delle costanti in SQL viene lasciata ai metodi `setXxxx(i, valore)`. Il programmatore Java infatti scriverà il parametro `valore` nella sintassi Java senza doversi preoccupare di come sia rappresentata la medesima costante in SQL.

Anche le transazioni possono essere realizzate attraverso le classi di JDBC, in particolare per attivare una transazione occorre disattivare la funzione di autocommit che risulta attiva per default. Ciò si ottiene attraverso un metodo dell'oggetto `con`:

```
con.setAutoCommit(false);
```

A questo punto vanno inviati al database server attraverso oggetti `Statement` o `PreparedStatement` i diversi comandi SQL e la transazione va poi conclusa con l'invocazione di uno dei seguenti metodi dell'oggetto `con`:

```
con.commit();
  oppure
con.rollback();
```

Alla fine della transazione per riattivare l'esecuzione di comandi SQL senza il meccanismo delle transazioni basta impostare correttamente la funzione di autocommit.

```
con.setAutoCommit(true);
```

Nella successiva sottosezione si mostra un esempio di pagina dinamica realizzata attraverso una servlet che interagisce via JDBC con un database server.

3.3 Esempio di servlet che realizza una pagina dinamica

In questa sezione si vuole mostrare una servlet completa che presenta l'elenco degli studenti estratti da una base di dati gestita su un database server Postgresql. Nella base di dati si suppone presente la seguente tabella: `STUDENTE(Matricola, Cognome, Nome, CreditiOttenuti)`. Si ipotizza di voler scrivere una servlet che, data la matricola di uno studente che arriva come parametro `matricola` dell'URL, sia in grado di estrarre i dati dello studente dalla base di dati e di spedire tali dati in una pagina web al browser.

```
import java.io.*;
import java.util.*;
import java.text.*; // per il DateFormat
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Query extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
          throws IOException, ServletException {

      String mat = request.getParameter("matricola");

      PrintWriter out = response.getWriter();
      String sql;
      Connection con = null;
      Statement stmt;
      ResultSet rs;

      // formattatore per le date
      DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT, Locale.ITALY);

      // parametri di connessione
      String url = "jdbc:postgresql://SERVER-POSTGRESQL/esercitazioni";
      String user = "";
      String passwd = "";

      try {
            Class.forName("org.postgresql.Driver");
          } catch (ClassNotFoundException cnfe) {
            System.err.println("Driver JDBC non trovato: "+cnfe.getMessage());
```

```
        }

        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"");
        out.println("           \"http://www.w3.org/TR/REC-html40/loose.dtd\">");
        out.println("<html>"); out.println("<head>");
        out.println("<title>Studenti</title>");
        out.println("</head>"); out.println("<body>");
        out.println("<h1>Studenti</h1>");

        try {
             con = DriverManager.getConnection(url,user,passwd);

             sql =  " SELECT * ";
             sql += " FROM STUDENTE ";
             sql += " WHERE matricola = '"+mat+"'";

             stmt = con.Statement();
             rs=stmt.executeQuery(sql);

             while (rs.next()) {
                    out.println("<p>");
                    out.println("<strong>Cognome:</strong> "+rs.getString("Cognome"));
                    out.println("<br>");
                    out.println("<strong>Nome:</strong> "+rs.getString("Nome"));
                    out.println("<br>");
                    out.println("<strong>Crediti ottenuti:</strong> "+
                                rs.getInt("CreditiOttenuti"));
                    out.println("</p>");
                    out.println("<hr>");
              }
              con.close();
        } catch (SQLException sqle) {
                System.err.println("DriverManager non trovato: "+sqle.getMessage());
        }
        out.println("</body>");
        out.println("</html>");
    }
}
```

Si noti che

- invece dei un'oggetto della classe `Statement` poteva essere usato un oggetto della classe `PreparedStatement`.
- il risultato dell'interrogazione poteva essere gestito anche senza il ciclo `while` in quanto l'interrogazione dovrebbe restituire una sola riga.

4 La realizzazione di una pagina web dinamica usando la tecnologia Java Server Pages di Sun

Prima di trattare nel dettaglio la realizzazione di pagine dinamiche con la tecnologia Java Server Pages (JSP) di Sun, presentiamo brevemente le caratteristiche di un *JSP engine*, vale a dire, del modulo software che viene attivato sul web server per consentire l'esecuzione di JSP.

L'attivazione di un *JSP engine* sul web server, richiede l'esistenza di un JavaServlet engine; solitamente il modulo unico ed consente contemporaneamente sia l'attivazione di servlet che l'attivazione di JSP. In particolare, un JSP engine è un servlet engine dove sono state attivate alcune classi Java aggiuntive in grado di gestire le richieste di attivazione di JSP. Si può dire che un JSP engine sia costituito da:

- un *Servlet Container*, che implementa una Java Virtual Machine (JVM), in grado di eseguire sevlet come descritto nella precedente sezione.
- una classe Java *Page Compiler Servlet* che si occupa della compilazione e attivazione di JSP.
- una classe Java per ogni JSP presente sul sito web.

Il funzionamento di un JSP engine è presentato in Figura 5. Si noti che l'arrivo di una richiesta HTTP al web server, la classe *Page Compiler Servlet*, verifica che la JSP non abbia subito modifiche e, se ha subito cambiamenti, rigenera la corrispondente servlet e la attiva, altrimenti passa direttamente all'attivazione della servlet precedentemente compilata. Come si può notare, per ogni JSP viene generato una corrispondente servlet e quindi la tecnologia delle Java Server Pages può essere vista come uno strumento che facilita la scrittura di servlet, ma che non ha rispetto a queste nessuno altro specifico vantaggio. La caratteristica principale delle JSP è quindi la sintassi adottata per la sua specifica. Tale sintassi prevede di immergere in un file HTML codice Java attraverso l'uso di marcatori specifici. Tali marcatori consentono di introdurre porzioni di codice Java in una qualsiasi posizione del file HTML, con l'effetto che, alla richiesta del file da parte del browser, tale codice verr sostituito dal risultato della sua esecuzione e si produrr quindi un puro file HTML.

Vediamo ora come si scrive una JSP.

4.1 I principali marcatori (tag) JSP

Come già detto, una JSP si ottiene da un file HTML introducendo particolari marcatori che corrispondono ai seguenti elementi JSP: *Directives* (o *Direttive*), *Declaration* (o *Dichiarazioni*), *Expressions* (o *Espressioni*) e *Scriptlet* (o *Codice Java*). Analizziamo ora ciascun elemento JSP.

- *Directive*: consente di specificare proprietà generali della servlet associata alla JSP, tali proprietà in particolare influiscono sulla compilazione della servlet associata alla JSP.
  Sintassi:

  ```
  <%@ NOME_DIRETTIVA ATTR_1 = "val_1" ... ATTR_n = "val_n" %>
  oppure
  <jsp:directive.NOME_DIRETTIVA ... />
  ```

  `NOME_DIRETTIVA` va sostituito con la direttiva che si vuole istanziare. Tra le direttive più usate troviamo la direttiva `page`, che presenta tra gli altri i seguenti attributi:

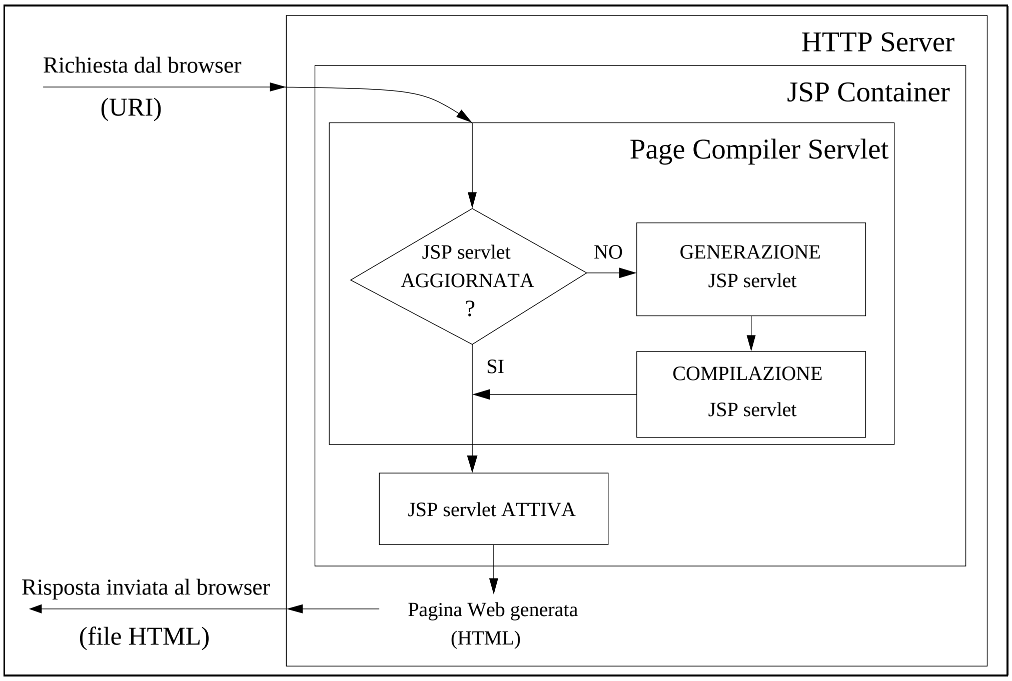

Figura 5: Architettura di base di un JSP engine.

- `import`: consente di precisare classi o package da importare per la compilazione;
- `errorPage`: consente di precisare l'URI di una pagina JSP da invocare in caso di errore;
- `isErrorPage` (valori possibili: `true` o `false`): indica se la pagina è una pagina di gestione dell'errore oppure no. Se impostato a `true` consente l'accesso all'oggetto `exception`.

Un'altra direttiva utile è la direttiva `include`, che presenta tra gli altri i seguenti attributi:

- `file`: consente di specificare l'URI del file da includere. Si noti che il file viene incluso prima di eseguire la compilazione della servlet associata alla JSP.

• *Declaration*: consente di definire variabili di classe e metodi statici da utilizzare in espressioni o scriplets contenuti nella pagina JSP.
Sintassi:

```
<%! DICHIARAZIONE %>
oppure
<jsp:declaration> DICHIARAZIONE </jsp:declaration>
```

`DICHIARAZIONE` va sostituito con la dichiarazione che si vuole inserire nella JSP e quindi nella servlet corrispondente. Il seguente è un possibile esempio di dichiarazione:

```
<%! private int x=0, y=1; %>
```

• *Expression*: consente di specificare un'espressione Java di tipo qualsiasi. All'esecuzione della JSP la marca che contiene l'espressione viene sostituita con una stringa che rappresenta il risultato della valutazione dell'espressione stessa.
Sintassi:

```
<%= ESPRESSIONE %>
oppure
<jsp:expression> ESPRESSIONE </jsp:expression>
```

`ESPRESSIONE` rappresenta l'espressione Java da calcolare. Il seguente è un possibile esempio di espressione:

```
<%!= x + y %>
```

- *Scriptlet*: consente di specificare porzioni di codice Java in un punto qualsiasi del file HTML. All'esecuzione della JSP la marca che contiene la porzione di codice Java viene sostituita con una porzione di codice HTML che rappresenta il risultato dell'esecuzione del codice Java.
  Sintassi:

  ```
  <% CODICE JAVA %>
  oppure
  <jsp:scriptlet> CODICE JAVA </jsp:scriptlet>
  ```

Si mostra ora un esempio di JSP che calcola il fattoriale dei primi 20 numeri interi positivi dove si usano: dichiarazioni, espressioni e scriptlet.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
  <title>
    Factorial loop
  </title>
</head>

<%-- Declaration: metodo per il calcolo del fattoriale --%>
<%! public long fact (long y) {
    if (x == 0) return 1;
    else return x * fact(x-1);
    }
%>

<body>
  <h1>Factorial loop: calcolo del fattoriale</h1>
<table>
<tr>
<th>X</th>
<th>X!</th>
</tr>
<%-- Scriptlet: inizio del ciclo --%>
<% for (long x=0; x<=20; ++x) { %>
<tr>
<td>
<%= x %>
</td>
```

```
<td>
<%= fact(x) %>
</td>
<%-- Scriptlet: fine ciclo --%>
<% } %>
</table>
</body>
</html>
```

Si sottolinea che ogni JSP viene sempre convertita in una servlet. Tale servlet viene costruita a partire dall'intero contenuto del file JSP, ciò significa che

- ogni elemento JSP (in particolare, dichiarazioni, espressioni e scriptlet) viene copiato nel metodo doGet (e doPost) della servlet e
- ogni riga HTML ... testoHTML ... viene riportata nella servlet e convertita in una riga del tipo:

  ```
  out.println("... testoHTMl ...");
  ```

Va precisato inoltre che, poiché la stesura di una JSP va sempre considerata come una macro per la scrittura di una servlet, nel codice Java contenuto nelle dichiarazioni, espressioni e scriptlet è possibile fare riferimento ad un insieme di oggetti implicitamente presenti. Tali oggetti non sono altro che gli oggetti usualmente presenti in una servlet, quindi, innanzitutto i parametri del metodo doGet (o doPost) request e response, ma anche l'oggetto out della classe PrintWriter e altri oggetti tra i quali, ad esempio, l'oggetto exception che corrisponde ad un oggetto della classe Throwable ed è disponibile nelle JSP dichiarate come pagine di gestione dell'errore (direttiva page attributo isErrorPage a true).

Infine, esiste una ulteriore classe di marcatori che consente di inserire nella JSP elementi denominati JSP action o azioni JSP. Tali elementi hanno diverse funzioni, tuttavia, per ciò che riguarda la generazione di pagine dinamiche tre sono i tipi di azioni che vogliamo presentare:

- <jps:forward page= localURL»: tale azione consente di trasferire il controllo ad un'altra JSP o servlet; è quindi di fondamentale importanza se si vuole organizzare l'applicazione web in modo modulare e trasferire il controllo durante l'esecuzione da un modulo all'altro.
- <jps:include page= localURL»: tale azione consente di includere nella pagina JSP corrente il codice HTML generato da un'altra JSP o servlet: anche questa azione è fondamentale per la gestione modulare dell'applicazione web.
- <jps:useBean ...>, <jsp:setProperty ...>, <jsp:getProperty>: tali azioni consentono di accedere e manipolare il contenuto di oggetti JavaBeans; come si illustrerà nell'ultima sezione della dispensa, la manipolazione di oggetti JavaBeans risulta estremamente utile nel caso in cui il risultato di un'interrogazione sia rappresentato attraverso un vettore (o array) di oggetti JavaBeans.

La realizzazione di una pagina dinamica usando una JSP risulta quindi molto più semplice rispetto al caso in cui si utilizzi una servlet. Infatti, mentre l'interazione con il database server avviene esattamente come avveniva in una servlet sfruttando le classi della libreria JDBC, la generazione dell'HTML statico viene ottenuta semplicemente e in modo automatico dal file HTML in cui si inseriscono i marcatori JSP.

Per mostrare le differenze tra una pagina dinamica scritta con una servlet e una pagina dinamica scritta con una JSP, si riporta di seguito la JSP che genera la pagina dinamica che nella sezione precedente è stata realizzata con una servlet.

4.2 Esempio di JSP che realizza una pagina dinamica

In questa sezione si vuole mostrare una JSP completa che presenta l'elenco degli studenti estratti da una base di dati gestita su un database server Postgresql. Nella base di dati si suppone presente la seguente tabella: `STUDENTE(Matricola, Cognome, Nome, CreditiOttenuti)`. Si ipotizza di voler scrivere una JSP che, data la matricola di uno studente che arriva come parametro `matricola` dell'URL, sia in grado di estrarre i dati dello studente dalla base di dati e di spedire tali dati in una pagina web al browser.

```
<%@ page import="java.io" %>
<%@ page import="java.util.Locale" %>
<%@ page import="java.text.DateFormat" %>
<%@ page import="java.sql" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd\">
<html>
<head>
<title>Studenti</title>
</head>
<%! String mat = request.getParameter("matricola");
          String sql;
          Connection con = null;
          Statement stmt;
          ResultSet rs;
          DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT, Locale.ITALY);

          String url = "jdbc:postgresql://SERVER-POSTGRESQL/esercitazioni";
          String user = "";
          String passwd = "";
      %>
      <% try {
            Class.forName("org.postgresql.Driver");
          } catch (ClassNotFoundException cnfe) {
            System.err.println("Driver JDBC non trovato: "+cnfe.getMessage());
      } %>
<body>
<h1>Studenti</h1>
      <% try {
               con = DriverManager.getConnection(url,user,passwd);
             } catch (SQLException sqle) {
               System.err.println("DriverManager non trovato: "+sqle.getMessage());
         }
         try {
               sql =  " SELECT * ";
               sql += " FROM STUDENTE ";
               sql += " WHERE matricola = '"+mat+"'";

               stmt = con.Statement();
               rs=stmt.executeQuery(sql);
             } catch (SQLException sqle) {
```

```
                System.err.println("Errore in interrogaizone SQL: "+sqle.getMessage());
            }
        %>
        <%-- INIZIO CICLO --%>
            while (rs.next()) {
        %>
                <p>
                  <strong>Cognome:</strong><%= rs.getString("nome")) %>
                  <br>
                  <strong>Nome:</strong><%= rs.getString("congnome")) %>
                  <br>
                  <strong>Crediti ottenuti:</strong><%= rs.getInt("crediti_ottenuti")) %>
                  </p>
                  <hr>
        <%  }
             con.close();
        %>
        <%-- FINE CICLO --%>
        </body>
</html>
```

Si noti che la stesura della JSP rispetto alla servlet risulta più semplice e più vicina all'impostazione sintattica di un usuale file HTML.

Infine, si fa notare che i commenti JSP sono compresi tra i marcatori `<%-- commento JSP --%>` e che tali commenti non vengono inviati al browser con il file HTML. Tale sorte spetta invece ai tradizionali commenti HTML racchiusi tra marcatori diversi: `<!-- commento HTML ->`.

5 Una metodologia di progettazione per siti web centrati sui dati

In questa sezione si presenterà una metodologia di progettazione di un sito web centrato sui dati. In particolare si vuole presentare allo studente un metodo per affrontare la progettazione di un sito di questa categoria, dove si individuino una serie di fasi da seguire durante la progettazione e, per ogni fase, si propongano alcuni criteri e strumenti di supporto alla progettazione.

Innanzittutto va chiarito cosa si intende per sito web centrato sui dati e, in particolare, quali siano gli elementi caratterizzanti che devono essere presenti per applicare la metodologia proposta.

Va precisato che con sito web si intende in questo modulo un insieme di pagine web *contenenti informazioni relative ad una specifico contesto e sotto il controllo di una singola organizzazione.* Ad esempio, costituiscono un sito web le pagine che descrivono le attività svolte da un'azienda, oppure le pagine che descrivono il dipartimento di un'università. Solitamente l'insieme di pagine che costituiscono un sito web sono organizzate in modo da essere tutte raggiungibili, attraverso uno o più link, da un unica pagina *radice* detta *home page* del sito.

5.1 Un sito web centrato sui dati

Come riconoscere un sito web centrato sui dati? In particolare, come riconoscere che il sito che vogliamo progettare è un sito centrato sui dati? Esistono in realtà due possibili situazioni in cui la soluzione più conveniente per la pubblicazione delle informazioni risulta essere un sito web centrato sui dati:

1. Nell'organizzazione, che si deve dotare del sito web, esistono uno o più sistemi che gestiscono una o più basi di dati contenenti la maggior parte delle informazioni da pubblicare nel sito.
2. L'informazione da presentare nel sito ha una struttura ben definita e abbastanza costante nel tempo, anche se non è ancora presente una base di dati che contiene tale informazione.

In entrambe i casi il sito da realizzare si occupa della presentazione di *informazione strutturata*, già presente in una base di dati, oppure da memorizzare in una nuova base di dati anch'essa da realizzare. La presenza di dati strutturati è caratteristica indispensabile per poter parlare di sito centrato sui dati.

Più precisamente possiamo affermare che un sito web centrato sui dati è un sito web che possiede le seguenti caratteristiche:

- molte pagine che costituiscono il sito hanno una struttura simile in quanto presentano lo stesso tipo di informazione. Ad esempio, le pagine web, che all'interno di un sito presentano le informazione relative ai docenti di un corso di studi, sono una per docente, ma hanno tutte la stessa struttura.
- l'informazione da pubblicare sul sito ha una struttura ben definita e quindi risulta rappresentabile in una base di dati.
- l'informazione da rappresentare ha un elevato tasso di aggiornamento, vale a dire, sono frequenti operazioni di inserimento, modifica o cancellazione di dati e ciò richiede il conseguente aggiornamento delle pagine web del sito.

Queste tre caratteristiche implicano che sia conveniente realizzare un sito web centrato sui dati adottando un sistema per la gestione di basi di dati (che chiamiamo database server) sul quale creare la base di dati che conterrà le informazioni da pubblicare. Se tale sistema esiste già, risulta evidente che il sito dovrà interagire con esso. Inoltre, le pagine del sito dovranno essere progettate in modo da risultare dinamiche nel contenuto: vale a dire, l'interazione tra web server e database server dovrà essere realizzata in tempo reale in modo che, ad ogni richiesta di una pagina web, il web server richieda al database server i dati necessari per completare la pagina web da rispedire al browser. Il sistema che vogliamo realizzare genera quindi pagine web dinamica e lavora secondo lo schema mostrato in Figura 6.

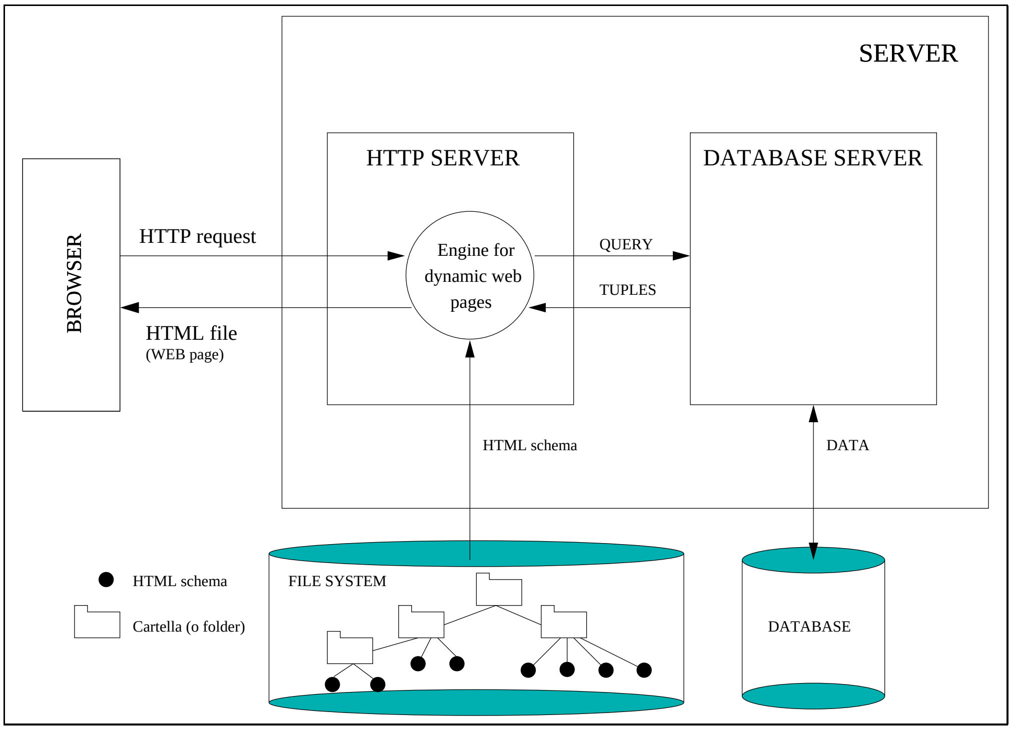

Figura 6: Generazione di una pagina web dinamica.

Si noti come una richiesta proveniente dal browser generi una interazione con il database server, per il recupero dei dati da pubblicare, e il successivo inserimento degli stessi in un file HTML che viene inviato al browser.

Vediamo ora di capire quali siano le fasi in cui possiamo strutturare la progettazione di un sito web centrato sui dati. Ipotizziamo che la base di dati che contiene le informazioni da pubblicare sia già presente e quindi non vada progettata. Ciò implica che sia già presente, in qualche forma, sia lo schema concettuale che ne descrive la struttura e il contenuto, sia lo schema logico, che rappresenta la realizzazione della base di dati sul database server. Tale schema logico verrà utilizzato in una delle fasi di progettazione del sito, in particolare, quando dovremo specificare da dove prelevare le informazioni da pubblicare nelle pagine web.

Se la base di dati non fosse presente, la sua progettazione costituirebbe la prima fase (FASE 0) dell'intero progetto del sito e obiettivo di tale fase sarebbe la specifica dello schema concettuale della base di dati stessa. Esso infatti rappresenta il *contenuto informativo* del sito e consente di produrre lo schema logico della base di dati, vale a dire la descrizione della struttura della base in uno specifico modello dei dati (solitamente il modello relazionale).

Ciò premesso, la progettazione di un sito web centrato sui dati ha come obiettivo quello di specificare lo schema logico del sito e tale specifica rappresenterà il documento di partenza per la successiva realizzazione del sito stesso. Lo schema logico di una sito centrato sui dati deve precisare:

1. *l'organizzazione in pagine web del contenuto informativo* del sito: va specificato quali sono le pagine presenti nel sito e quali dati contengono. Si noti che ciò non significa precisare esattamente il contenuto di ogni pagina web del sito, ma, viste le caratteristiche dell'informazione da pubblicare (dati strutturati), significa specificare solo il contenuto di ogni *tipo di pagina*. Ad esempio, non va specificato il contenuto di tutte le pagine web che descrivono docenti, ma soltanto il contenuto di una generica pagina che descrive un docente (tipo di pagina docente), indicando quali attributi, che descrivono un docente, verranno mostrati (nome, cognome, ecc...) senza precisare le informazioni stesse (rispettivamente, Mario, Rossi, ecc...).

2. *le modalità di navigazione tra le pagine web del sito*: va specificato quali sono i link logici tra le pagine e quali ulteriori link di supporto alla navigazione tra le pagine devono essere realizzati.

3. *la corripondenza tra il contenuto delle pagine e la base di dati*, vale a dire, come i dati da presentare nella pagina vengono estratti dalla base di dati.

4. *la presentazione delle infomazioni nelle pagine* del sito: va specificato per ogni pagina presente nel sito il suo aspetto grafico e la disposizione del contenuto informativo e dei link.

Per ciascuno dei punti in cui si articola lo schema logico del sito è possibile prevedere una fase nella metodologia di progettazione e la sequenza di tali fasi corrisponde all'ordine di presentazione dei punti sopra elencati. Per chiarire il ruolo di ciascuna di queste fasi vediamo il seguente esempio.

Esempio SITO AZIENDALE

Supponiamo di considerare i dati relativi ai progetti in corso di svolgimento presso le varie sedi di un'azienda e di avere a disposizione una base di dati che contiene le informazioni da pubblicare. Supponiamo che lo schema concettuale della base di dati sia quello riportato in Figura 7.

La FASE 1 del progetto corrisponde alla specifica dei tipi di pagine presenti e del loro contenuto. In questo caso il sito potrebbe essere organizzato come segue:

- una home-page contenente la lista delle sedi dell'azienda, di cui si riporta il nome e la città.
- una pagina per ogni sede contenente: il nome e l'indirizzo della sede, il nome e cognome del direttore e l'elenco dei progetti attivi presso la sede.
- una pagina per ogni progetto contenente: i dati descrittivi del progetto, il responsabile e l'elenco dei dipendenti partecipandi al progetto.
- una pagina per ogni dipendente contenente: il nome, il cognome, lo stipendio e la qualifica del dipendente, la sede di lavoro e l'elenco dei progetti a cui partecipa.
- una pagina contenente l'elenco dei dipendenti dell'azienda, dove si mostrano il nome e il cognome dei dipendenti e la sede in cui lavorano.
- una pagina contenente l'elenco dei progetti attivi dell'azienda, dove si mostrano il nome del progetto e il cognome del responsabile.
- una pagina contenente l'elenco dei dipendenti di una sede dell'azienda organizzata come la pagina che mostra l'elenco di tutti i dipendenti dell'azienda.

La FASE 2 del progetto corrisponde alla specifica per ogni tipo di pagina dei link logici presenti e dei link presenti in tutte le pagine per facilitare la navigazione. Ad esempio, in questo caso:

- nella home-page si prevede la presenza di un link per ogni sede dell'azienda, un link verso la pagina contenente l'elenco dei dipendenti dell'azienda e un link verso la pagina contenente l'elenco dei progetti dell'azienda.
- nella pagina che descrive una sede dell'azienda si prevede: un link verso la pagina che descrive il direttore e, per ogni progetto attivo presso la sede, un link verso la pagina che descrive il progetto stesso.
- nella pagina che descrive un progetto si prevede: un link verso la pagina che descrive il responsabile e un link per ogni dipendente che partecipa al progetto verso la pagina che lo descrive.
- nella pagina che descrive un dipendente si prevede un link verso la pagina che descrive la sede di lavoro e un link per ogni progetto a cui il dipendente partecipa verso la pagina che descrive il progetto stesso.
- nella pagina contenente l'elenco dei dipendenti dell'azienda si prevede un link per ogni dipendente verso la pagina che lo descrive.
- nella pagina contenente l'elenco dei progetti attivi dell'azienda si prevede un link per ogni progetto verso la pagina che lo descrive.
- nella pagina contenente l'elenco dei dipendenti di una sede dell'azienda un link per ogni dipendente verso la pagina che lo descrive.

Inoltre potremmo prevedere in ciascuna pagina alcuni link di supporto alla navigazione come ad esempio un link verso la pagina web precedente rispetto a quella corrente oppure una sequenza di link che rappresenta il percorso logico seguito per arrivare a questa pagina dalla home page del sito (ad esempio se siamo sulla pagina di un progetto potremmo fornire la seguente sequenza di link: HOME-PAGE → LISTA-PROGETTI).

Nella FASE 3 va precisato come ottenere dalla base di dati le informazioni da pubblicare nelle pagine. Questo si può specificare indicando, per ogni dato o insieme di dati da pubblicare, l'interrogazione da eseguire sulla base di dati per ottenerli, secondo quanto descritto nello schema logico della base di dati stessa. L'interrogazione si può ad esempio scrivere in SQL, se questo è il linguaggio di interrogazione adottato dal database server che gestisce la base di dati (come accade nella maggior parte dei casi).

Ad esempio per la home page, se supponiamo di generare l'elenco delle sedi indicandone il nome e la città dove si trovano, potremmo precisare che tale elenco viene prodotto dalla seguente interrogazione SQL:

```
SELECT nome_sede, citta FROM Sede
```

Sede è infatti una tabella della base di dati relazionale ottenuta dalla traduzione della schema concettuale di Figura 7. La stessa cosa dovrebbe essere fatta per ogni tipo di pagina precisata nella FASE 1. Si noti che non sempre sarà sufficiente una interrogazione: in generale una pagina potrà richiedere una o più interrogazioni sulla base di dati.

L'ultima fase (FASE 4) prevede la specifica della presentazione, vale a dire, la descrizione di come si dispongono le informazioni sulla pagina, di quale sia il formato da adottare per il testo dell'informazione

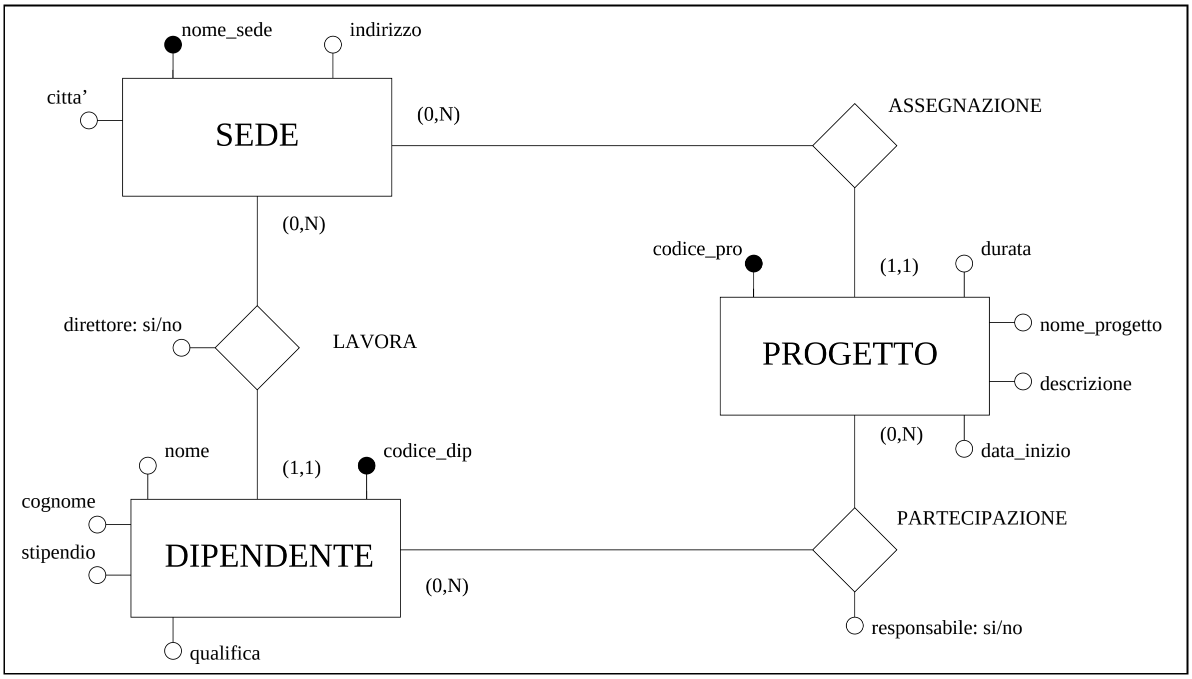

Figura 7: Schema concettuale della base di dati utilizzata nell'esempio SITO AZIENDALE.

e di quale sia l'aspetto grafico (immagini e altro) da impostare su ogni pagina. In questa dispensa la definizione della presentazione non viene approfondita.

Esercizi proposti

1. Si consideri lo schema concettuale mostrato in Figura 8 e si progetti, come fatto nell'esempio **SITO AZIENDALE**, un sito web che presenti le informazioni contenute nella base di dati descritta dallo schema. In particolare si scriva un documento che riporti una descrizione a parole, secondo la metodologia proposta, dei due seguenti aspetti: a) *l'organizzazione in pagine web del contenuto informativo* (FASE 1) e b) *le modalità di navigazione web del sito* (FASE 2) Lo schema concettuale presentato in Figura 8 modella le informazioni relative alla gestione di un corso di formazione composto da diversi insegnamenti. Supponiamo che ogni insegnamento abbia un unico docente e che gli studenti frequentino i diversi insegnamenti nei diversi anni in cui il corso è strutturato. Alla fine del corso la prova di verifica produce per ogni studente una valutazione (voto finale).

5.2 Un modello per la specifica del contenuto e della struttura delle pagine di un sito centrato sui dati

Oltre alla suddivisione in fasi la metodologia che si vuole presentare allo studente prevede uno strumento per la specifica dello schema logico delle pagine del sito. Tale strumento è il modello per la specifica di ipertesti orientati ai dati proposto da Atzeni, Ceri, Paraboschi e Torlone in [ACPT99] (si veda anche [ACPT01]). In tale modello un ipertesto, che nel nostro caso è l'insieme di pagine web che costituisce il sito, viene descritto specificando lo schema di ogni documento (pagina) che compone l'ipertesto, indicando i dati presenti nella pagina e gli eventuali link verso altre pagine. In questa sezione consideriamo

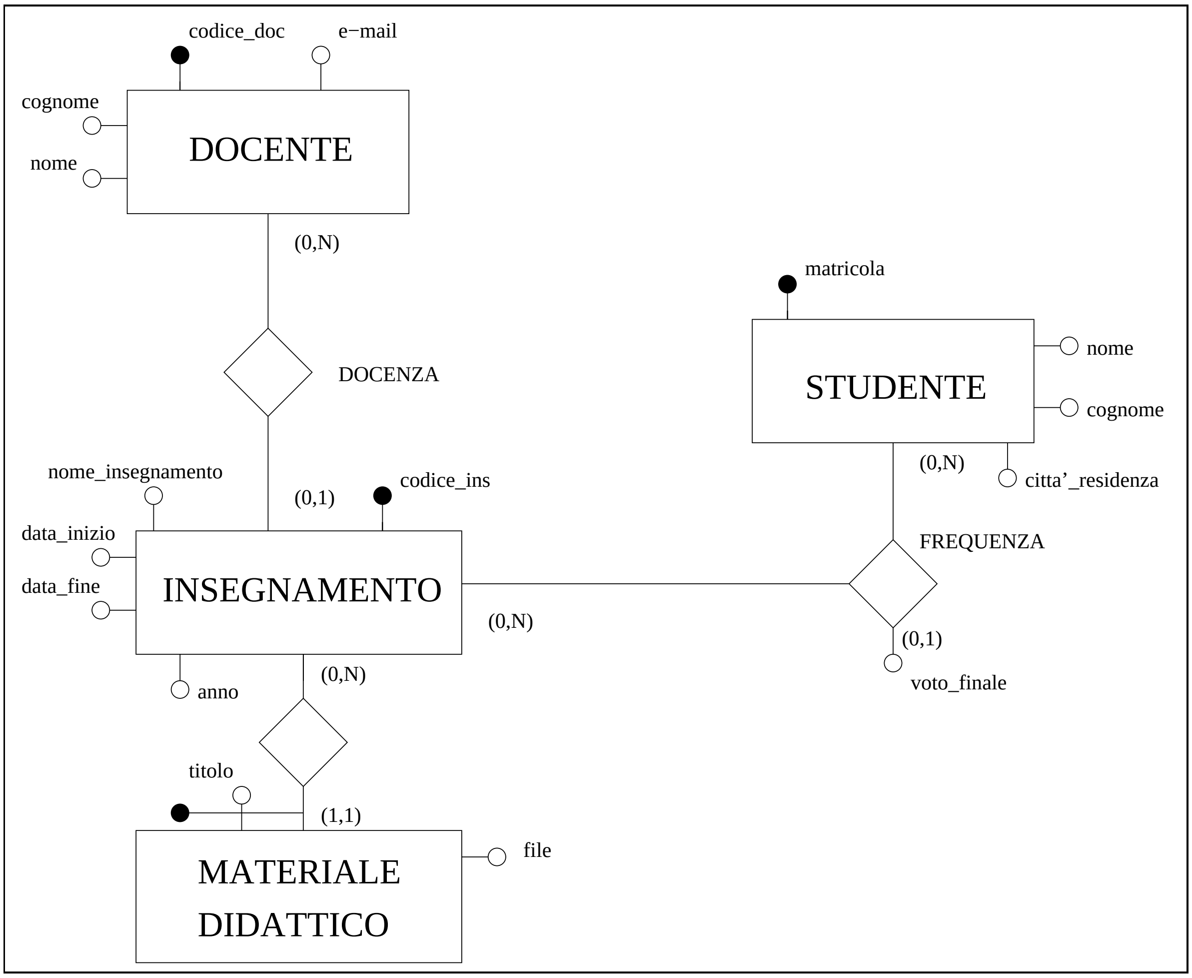

Figura 8: Schema concettuale relativo alla base di dati di un corso di formazione.

i costrutti del modello che ci consentono di precisare i dati presenti nelle pagine, nella prossima sezione si presenterà invece quella parte del modello che permette di aggiungere i link.

Vediamo ora la sintassi dei costrutti del modello. Ciò significa presentare le regole che il modello impone per la descrizione dello schema logico di un ipertesto.

Il costrutto base del modello che consente di specificare uno schema di pagina può essere istanziato nello schema logico dell'ipertesto secondo la seguente sintassi:

```
page-schema <nome_schema_pagina>
(
<nome_attributo_1> : <TIPO>;
....
<nome_attributo_n> : <TIPO>;
)

<TIPO> := {string | integer | date | real | <LISTA>}
```

```
<LISTA> := list-of ( <nome_attributo_1> : <TIPO>; ...
           <nome_attributo_n> : <TIPO>; )
```

Nei prossimi paragrafi viene spiegato il significato di questa definizione. Va innanzitutto precisato che specificare la sintassi di un costrutto del nostro modello significa indicare in quale modo è possibile scrivere un'istanza del costrutto stesso nel documento che rappresenta lo schema logico dell'ipertesto. Se vogliamo fare un paragone con la definizione dello schema logico di una base di dati, la definizione nello schema logico di un ipertesto di uno schema di pagina con il costrutto `page-schema` corrisponde alla definizione nello schema logico di una base di dati relazionale di uno schema di tabella con il costrutto `tabella o relazione` del modello relazionale .

Veniamo ora alla descrizione della sintassi del costrutto `page-schema`. La sintassi ci dice che ogni istanza di questo costrutto inizia con la parola chiave `page-schema` e prosegue con il nome che vogliamo dare allo schema di pagina e con una lista di attributi separati dal simbolo ; e racchiusi tra parentesi tonde. Per convenzione ciò che è racchiuso tra parentesi acute (ad esempio, $< X >$) va sostituito nell'istanza secondo la seguente regola:

- se X è scritto minuscolo, $< X >$ va sostituito con una stringa di caratteri (vale a dire solitamente una parola che rappresenta un nome), ciò accade nel nostro caso per <nome_schema_pagina>, <nome_attributo_1> e <nome_attributo_n>.
- se invece X è tutto maiuscolo, allora $< X >$ deve essere sostituito con quanto indicato nelle regole della sintassi, ciò accade nel nostro caso per <TIPO> e <LISTA>.

Nella definizione di <TIPO> si dice che tutte le volte che compare <TIPO> esso può essere sostituito con una delle seguenti parole chiave: `string`, `integer`, `date` o `real` oppure con un'istanza di <LISTA>. Infatti una sequenza di simboli racchiusi tra parentesi graffe e separati dal carattere | indica un'alternativa. Infine la definizione di <LISTA> ci dice che le istanze di lista iniziano con la parola chiave `lista-of` e proseguono con una sequenza di attributi separati dal simbolo ;.

Per chiarire tali regole sintattiche si mostra di seguito un esempio di schema per la pagina che presenta i dati di un dipendente dell'esempio SITO AZIENDALE.

```
page-schema Dipendente
(
Nome : string;
Cognome: string;
Stipendio: integer;
Qualifica: string;
Nome_Sede: string;
Citta'_Sede: string;
Progetti: list-of ( Nome_Progetto: string; );
)
```

Vediamo ora quale è il significato di tale definizione. Abbiamo definito lo schema delle pagine che rappresentano i dipendenti dell'azienda. Tale schema precisa che ogni pagina di questo tipo contiene 7 attributi che rappresentano: il nome, il cognome, lo stipendio, la qualifica del dipendente, il nome e la città della sede in cui il dipendente lavora e i progetti a cui partecipa. Inoltre per ogni attributo è stato specificato il tipo di valori che esso assume, così Nome, Cognome, Qualifica, Nome_sede e Città_Sede sono attributi di tipo `string`, che corrisponde alle stringhe di caratteri, l'attributo Stipendio è invece di tipo `integer`, che corrisponde ai numeri interi, e infine, l'attributo Progetti rappresenta la lista dei progetti a cui il dipendente partecipa attraverso una lista di elementi contenenti un attributo di tipo stringa di caratteri che rappresenta il nome del progetto. Altri tipi possibili non utilizzati in questo esempio sono: `date` e `real` che rappresentano rispettivamente le date e i numeri reali. Si noti che ogni attributo assume un sol valore se il suo tipo è `string`, `integer`, `real` o `date`, mentre assume una lista di valori nel caso in cui il suo tipo sia `list of`.

Va sottolineato che il `page-schema` Dipendente descrive un tipo di pagina: ciò significa che più di una pagina del sito avrà questo schema. In particolare, tutte le pagine del sito che descriveranno dipendenti avranno questo schema e nel sito che vogliamo realizzare esisterà una pagina per ogni dipendente, quindi molte pagine avranno questo schema. Tuttavia ciò non vale in generale per tutti gli schemi di pagina. Ad esempio, lo schema di pagina che descrive la home-page o lo schema di pagina che descrive l'elenco dei dipendenti dell'azienda avranno nel sito un'unica istanza, vale a dire, una sola pagina del sito avrà lo schema home-page e una sola lo schema dell'elenco dei dipendenti dell'azienda. Vediamo sintatticamente come specificare lo schema dell'elenco dei dipendenti dell'azienda.

```
page-schema Elenco_dipendenti unique
(
Dipendenti: list-of ( Cognome: string; Nome: string; );
)
```

Rispetto alla sintassi precisata precedentemente in questa istanza del costrutto `page-schema` abbiamo introdotto una parola chiave nuova: la parola `unique`. Tale parola chiave sta ad indicare che questo schema di pagina potrà avere nel sito una e una sola istanza: vale a dire, ci sarà una sola pagina con l'elenco dei dipendenti. L'unico attributo presente in questo schema rappresenta la lista dei dipendenti di cui si riporta il nome e il cognome.

Esercizi proposti

1. Si completi la rappresentazione nel modello proposto degli schemi di pagina descritti nell'esercizio **SITO AZIENDALE** della precedente sezione. Si presenti lo schema logico completo unendo in un unico documento le definizioni di `page-schema` presentate nel testo con quelle da voi aggiunte (non rappresentare i link, ma solo il contenuto informativo).

2. Si consideri il risultato dell'esercizio proposto nella sezione precedente e si specifichi attraverso il costrutto `page-schema` *l'organizzazione in pagine web del contenuto informativo* del sito (corso di formazione) (tralasciando ovviamente i link, per ora non rappresentabili nel modello).

5.3 Un modello per la specifica della struttura di navigazione di un sito centrato sui dati

Nella sezione precedente abbiamo visto come utilizzare il modello proposto da Atzeni, Ceri, Paraboschi e Torlone in [ACPT99], per la definizione dell'organizzazione in pagine web del contenuto informativo di un sito centrato sui dati. Ora è necessario completare la presentazione di tale modello con introduzione di un'ulteriore estensione della sintassi del costrutto `page-schema`: tale estensione ci consente di rappresentare i *link* presenti nelle pagine e quindi di descrivere la struttura di navigazione del sito.

Nelle pagine web i link costituiscono uno degli elementi caratterizzanti, in quanto consentono di trasformare il puro testo in *ipertesto* realizzando i legami tra le varie parti in cui è suddiviso il testo stesso. I link rappresentano quindi quei legami che consentono una scansione non sequenziale dell'ipertesto (navigazione). La specifica dei link presenti in una pagina rappresenta un punto fondamentale della progettazione, poichè consente di precisare quale sarà la struttura di navigazione del sito.

Per specificare i link nello schema logico del sito estendiamo la sintassi del costrutto `page-schema` come segue:

```
page-schema <nome_schema_pagina>
(
<nome_attributo_1> : <TIPO> ;
....
<nome_attributo_n> : <TIPO> ;
)
<TIPO> := {string | integer | date | real | <LISTA> | <LINK>}
<LISTA> := list-of ( <nome_attributo_lista_1> : <TIPO>; ...
           <nome_attributo_lista_n> : <TIPO>; )
<LINK> := { link ( <ETICHETTA>; *<nome_altro_schema_pagina> ) |
            link ( <ETICHETTA>; url ( <nome_url> ) ) |
            link ( <ETICHETTA>; url ( <nome_url> , <nome_attributo> ) ) }
<ETICHETTA> := { <etichetta_costante> |
                 <nome_attributo>: { string | integer | date } }
```

Come si nota, a differenza della definizione data nella precedente sezione, è stato aggiunto il tipo `link` come tipo possibile per gli attributi dello schema di pagina. Il tipo link, associato ad un attributo, indica che l'attributo stesso rappresenta un link verso un'altro schema di pagina, descritto da uno specifico `page-schema`; in particolare, il tipo link consente di specificare:

- un'etichetta (la cui sintassi viene descritta da <ETICHETTA>): che può essere una parola costante (è uguale su tutte le istanze dello schema) oppure un attributo (il suo valore può cambiare da un'istanza all'altra dello schema), e
- il nome del `page-schema` a cui il link punta: è rappresentato sintatticamente dal nome del `page-schema` preceduto da un asterisco ('*').

oppure la struttura del link può essere più semplice, ciò accade quando non si intende puntare ad un `page schema` ma si intende puntare ad un URL costante esterno al sito web che si sta progettando: in tal caso la sintassi richiede di inserire la parola chiave `url` seguita dalla specifica dell'URL. Infine, se l'URL da specificare non è costante ma dipende dall'istanza oppure, essendo contenuta in una lista, dipende dall'elemento della lista (ad esempio, quando l'URL rappresenta il nome di un file scaricabile), la sintassi richiede di specificare, oltre alla parola chiave `url`, l'attributo in base al quale calcolare l'URL per ogni istanza. L'attributo viene precisato dopo la stringa che rappresenta la parte fissa dell'URL ed è separata da questa con una virgola.

Vediamo un esempio di `page-schema` contenente un attributo di tipo `link`. Con riferimento all'esempio **SITO AZIENDALE** della sezione precedente, supponiamo di voler arricchire lo schema di pagina `Dipendente`, che presenta i dati di un dipendente, con un link verso lo schema di pagina che descrive una sede, in particolare la sede in cui il dipendente lavora. Ciò si può ottenere dichiarando di tipo `link` l'attributo `Nome_Sede` che già era stato inserito in precedenza. La dichiarazione del `page-schema` Dipendente risulterà modificato nel seguente modo:

```
page-schema Dipendente
```

```
(
Nome : string;
Cognome: string;
Stipendio: integer;
Qualifica: string;
Nome_Sede: link (Nome: string; *Sede);
Citta'_Sede: string;
Progetti: list-of ( Nome_Progetto: string; );
)
```

Si noti che l'attributo `Nome_sede`, presente nello schema, produce come effetto sulle pagine web, istanze dello schema, la visualizzazione del nome della sede dove l'impiegato lavora e la generazione, in corrispondenza di tale nome, di un link verso la pagina, istanza del `page-schema` Sede, che descrive la sede stessa.

È possibile dichiarare di tipo `link` anche un attributo contenuto in una lista. Ad esempio, nel precedente schema è possibile dichiarare di tipo `link` anche l'attributo `Nome_Progetto` presente nella lista `Progetti`. In tal modo, nella pagina web che descrive un impiegato, viene mostrata la lista dei progetti a cui partecipa e, in corrispondenza del nome del progetto, viene generato un link verso la corrispondente pagina web che lo descrive. Lo schema finale risulta quindi essere definito come segue:

```
page-schema Dipendente
(
Nome : string;
Cognome: string;
Stipendio: integer;
Qualifica: string;
Nome_Sede: link (Nome: string; *Sede);
Citta'_Sede: string;
Progetti: list-of (Progetto: link(Nome_Progetto: string; *Progetto););
)
```

Come esempio di link con etichetta costante possiamo supporre di considerare la pagina iniziale del nostro sito (*home page*). In tale pagina sarà presente ad esempio un link verso la pagina che riporta l'elenco dei progetti dell'azienda. La specifica di tale link nel `page-schema` della home page risulterà essere il seguente:

```
page-schema Home_Page unique
(
....
Progetti: link ( "Progetti attivi presso la nostra azienda";
                 *Lista_Progetti);
....
)
```

In tale schema si suppone che sia già stato definito lo schema di pagina per la lista dei progetti con il nome `Lista_Progetti`.

Esercizi proposti

1. Si completi l'esercizio della precedente sezione aggiungendo allo schema logico del **SITO AZIENDALE** la specifica dei link.
2. Si consideri il risultato dell'esercizio proposto nella sezione precedente relativo al sito del corso di formazione e si completi lo schema logico aggiungendo i link.

5.4 Corrispondenza tra contenuto delle pagine e base di dati

Una volta specificato lo schema logico del sito, otteniamo un documento che descrive il contenuto informativo e la struttura di navigazione del sito che vogliamo realizzare. A questo punto occorre completare la documentazione del progetto logico aggiungendo la specifica delle corrispondenze tra le informazioni pubblicate nelle pagine del sito e la base di dati, che contiene e gestisce le informazioni stesse. Come già indicato precedentemente, supponiamo che la progettazione concettuale e logica della base di dati sia già avvenuta e, quindi, supponiamo che sia disponibile lo schema logico della base di dati. Tale schema, come già ipotizzato precedentemente, è uno schema relazionale. Per l'esempio **SITO AZIENDALE** lo schema relazionale della base di dati è di seguito riportato:

```
Sede(nome_sede, citta, indirizzo)
Dipendente(codice_dip, cognome, nome, stipendio, qualifica,
           sede, direttore)
Progetto(codice_pro, nome_progetto, descrizione,
         data_inizio, durata, nome_sede)
Partecipazione(codice_dip, codice_pro, responsabile)

CHIAVI PRIMARIE:
Sede: {nome_sede}
Dipendente: {codice_dip}
Progetto: {codice_pro}
Partecipazione: {codice_dip, codice_pro}
```

Per specificare la corrispondenza con la base di dati è necessario analizzare i `page-schema` definiti nello schema logico del sito e, per ogni attributo specificato, indicare come questo si ottiene dalla base di dati. Nel fare ciò, occorre innanzitutto individuare, per ogni `page-schema`, le tabelle della base di dati dalle quali vanno estratti i dati in esso visualizzati. Può accadere che i dati provengano da una sola tabella, oppure anche da più tabelle. Individuate le tabelle vanno infine specificate le interrogazioni SQL che consentono di estrarre i dati significativi per il `page-schema`.

Vediamo ora come procedere per la definizione delle interrogazioni SQL associate a un `page-schema` dello schema logico del sito. Due sono le caratteristiche di un `page-schema` che influiscono su tale definizione:

- il fatto che il `page-schema` sia di tipo `unique` oppure no;
- il fatto che gli attributi in esso contenuti siano di tipi semplici(`string`, `integer`, `date` e `real`) oppure di tipo `list-of` o di tipo `link`.

Consideriamo innanzitutto la situazione in cui il `page-schema` sia di tipo `unique`. In questo caso esiste una sola istanza per questo tipo di pagina: quindi è possibile estrarre dalla base di dati le informazioni da inserire in questa unica istanza attraverso una interrogazione SQL *costante*. Vale a dire, l'interrogazione

SQL non dipende dalla richiesta dal browser, ma resta sempre la stessa tutte le volte, visto che l'istanza di questo `page-schema` è una sola.

Ad esempio, consideriamo la seguente porzione di schema relativo alla prima pagina del sito specificato nell'esempio **SITO AZIENDALE**:

```
page-schema Home_Page unique
(
Sedi: list-of ( Sede: link (Nome_sede: string; *Sede); );
Progetti: link ( "Progetti attivi presso la nostra azienda";
                 *Lista_Progetti);
....
)
```

L'attributo Sedi presenta la lista delle sedi dell'azienda e l'elenco dei suoi valori si ottiene attraverso la seguente interrogazione SQL (come già visto in precedenza):

```
SELECT nome_sede FROM Sede
```

Ad ogni richiesta della Home_Page, il sistema esegue l'interrogazione sopra esposta e inserisce il risultato nella pagina web che viene restituita al browser, in tal modo, se viene aggiunta una sede all'azienda, non appena questa viene inserita nella base di dati verrà anche visualizzata sul sito.

La situazione cambia se il `page-schema` non è di tipo `unique`. Infatti, in questo caso esistono più pagine web che sono istanze dello stesso `page-schema` e, quindi, l'interrogazione SQL deve necessariamente cambiare tutte le volte. Ad esempio, consideriamo la seguente porzione dello schema della pagina web che descrive un dipendente, sempre con riferimento al sito specificato nell'esempio **SITO AZIENDALE**:

```
page-schema Dipendente
(
Nome : string;
Cognome: string;
Stipendio: integer;
Qualifica: string;
....
)
```

L'idea è che, in fase di realizzazione del sito, si generi un solo file HTML per definire la presentazione della pagina di un dipendente, e che sia poi l'interrogazione a selezionare, di volta in volta, i dati corretti per completare il file HTML a seconda di quale sia la pagina richiesta dal browser. Come è possibile che il server web sappia quale interrogazione fare, se viene richiesto sempre lo stesso `page-schema`? Per risolvere questo problema si introduce nella richiesta della pagina un certo numero di *parametri*. Si tratta di informazione aggiuntiva, accodata alla richiesta della pagina, che fornisce al server web ulteriori dettagli circa il servizio richiesto dal browser; nel nostro caso i parametri specificano quale istanza di un `page-schema` viene richiesta. Saranno questi parametri ad essere utilizzati dal server web per completare l'interrogazione SQL con cui estrarre l'informazione corretta dalla base di dati. I parametri, secondo il protocollo HTTP, possono essere specificati direttamente nella richiesta (detta anche URL - Universal Resource Locator) subito dopo il nome del servizio (pagina web) secondo la seguente sintassi:

```
http://www. ..... /<nome_schema_pagina>?
                   <nome_parametro_1>=<valore_parametro_1>&....
                   &<nome_parametro_n>=<valore_parametro_n>
```

Si noti che la lista di parametri, se presente, viene preceduta dal simbolo ?, inoltre, ogni parametro viene specificato indicandone il nome e il valore, infine, la specifica di un parametro è separata dalla specifica del successivo dal simbolo &. È possibile anche che i parametri vengano inviati dal browser al server in modo nascosto senza cioè comparire nell'URL. In tal caso la richiesta inviata e di tipo POST (vedi protocollo HTTP `http://www.w3c.org/Protocols`), mentre nel caso precedente è di tipo GET.

Tornando alla specifica delle interrogazioni associate ad un `page-schema`, per tutti gli schemi non `unique` diventa necessario, per quanto sopra osservato, definire uno o più parametri che consentano di individuare l'istanza richiesta. Nel caso del `page-schema Dipendente` è necessario introdurre un parametro per individuare il singolo dipendente. Tale parametro va scelto in modo che ci consenta di individuare i dati da pubblicare. Nel caso che stiamo considerando, potrebbe essere scelto un parametro che rappresenti l'attributo codice_dip presente nella tabella Dipendente (vedi lo schema logico della base di dati). Tale codice infatti individua univocamente un dipendente nella base di dati. A questo punto, ipotizzata la presenza del parametro *cod_dip* nella richiesta http, l'interrogazione SQL da eseguire può essere scritta come segue:

```
SELECT nome, cognome, stipendio, qualifica
FROM Dipendente
WHERE codice_dip = ?cod_dip?
```

Si noti che nell'interrogazione SQL è stato inserito il parametro `cod_dip` racchiuso tra due simboli ?. Non si tratta ovviamente di sintassi standard SQL, ma di una convenzione che stabiliamo ora per la specifica delle interrogazioni SQL associate ad un `page-schema`.

Descrivendo da capo l'interazione tra browser e http server per la richiesta della pagina web che descrive un singolo dipendente, si possono evidenziare i seguenti passaggi:

- Il browser richiede la pagina di uno specifico dipendente: ciò è quanto accade ad esempio quando, sulla pagina correntemente visualizzata dal browser, l'utente clicca il link di un certo dipendente; supponendo in particolare che l'utente clicchi sul link del dipendente con codice_dip uguale a 4, si genera la seguente richiesta:

  ```
  http://www. .../Dipendente?cod_dip=4
  ```

  Si noti che nella richiesta è presente il parametro `cod_dip` e che a tale parametro è stato assegnato il valore 4 in quanto l'utente, in questa simulazione, ha esattamente cliccato sul link relativo al dipendente con codice=4. Si sottolinea che l'URL sopra riportata era già presente nella pagina web visualizzata dal browser completa del parametro e del suo valore. Nel file HTML che rappresenta la pagina web attuale era presente cioè il seguente link:

  ```
  <a href="http://www. .../Dipendente?cod_dip=4">Mario Rossi</a>
  ```

- Quando la richiesta viene ricevuta dal http server essa viene risolta attivando un certo insieme di interrogazioni sul database server dove al posto degli eventuali parametri presenti nelle interrogazioni viene messo il valore ricevuto nella richiesta arrivata dal browser. In particolare, per ottenere gli attributi nome, cognome, stipendio e qualifica verrà attivata la seguente interrogazione:

```
SELECT nome, cognome, stipendio, qualifica
FROM Dipendente
WHERE codice_dip = 4
```

dove al parametro `cod_dip` è stato sostituito il valore 4.

- a questo punto il risultato dell'interrogazione verrà integrato dal http server con il file HTML che rappresenta lo schema di presentazione dei dipendenti e il file HTML risultante verrà restituito al browser.

Consideriamo ora il caso in cui un attributo del `page-schema` sia di tipo `list-of`. Per questo attributo sarà necessario estrarre dalla base di dati l'insieme di valori che appartengono alla lista: ciò significa che deve essere eseguita una query specifica per tale attributo, che si aggiunge ad altre interrogazioni eventualmente già associate al `page-schema` per estrarre i valori di altri attributi.

Infine se un attributo è di tipo `link` sarà necessario estrarre dalla base di dati le informazioni che ci consentono di ricostruire l'etichetta del link e, insieme a queste, anche le informazioni che ci consentono di definire il link stesso. Ciò significa estrarre i valori dei parametri che eventualmente vanno specificati nella richiesta di pagina che va inserita nel link (come visto sopra per l'esempio sul dipendente con `cod_dip` uguale a 4).

Ad esempio, consideriamo lo schema di pagina completo che descrive un dipendente:

```
page-schema Dipendente
(
Nome : string;
Cognome: string;
Stipendio: integer;
Qualifica: string;
Nome_Sede: link (Nome: string; *Sede);
Citta'_Sede: string;
Progetti: list-of ( Progetto: link ( Nome_Progetto: string; *Progetto); );
)
```

Quali interrogazioni devono essere eseguite per estrarre tutti i dati necessari a tale schema? Per rispondere a questa domanda occorre innanzitutto verificare quali insiemi di dati devono essere forniti per valorizzare (dare un valore a) tutti gli attributi dello schema. Va ovviamente considerato lo schema logico della base di dati dell'esempio **SITO AZIENDALE**. In tale schema notiamo che:

- Gli attributi `Nome`, `Cognome`, `Stipendio` e `Nome_sede` sono presenti nella tabella Dipendente e possono quindi essere estratti da questa; inoltre, l'attributo `Citta'_sede` è presente solo nella tabella Sede e quindi sarà necessario eseguire un join tra la tabella Dipendente e la tabella Sede per ottenere anche questo attributo;

- L'attributo `Progetti` è di tipo `list-of` e quindi va definita per esso una interrogazione specifica; tale interrogazione estrarrà il nome del progetto a cui il dipendente partecipa dal risultato del join tra la tabella Partecipazione (che rappresenta la relazione concettuale tra l'entità Progetto e l'entità Dipendente) e la tabella Progetto. Inoltre essendo tale attributo una lista di link l'interrogazione dovrà estrarre anche i dati necessari alla definizione del link stesso: in questo caso il codice del progetto (attributo codice_pro della tabella Partecipazione o della tabella Progetto).

- Si noti che, per quanto riguarda l'attributo `Nome_sede`, il fatto che sia di tipo `link` richiederebbe di estrarre i dati necessari per la definizione del link stesso, tuttavia, poiché il parametro necessario per definire tale link supponiamo sia il nome della sede, non deve essere estratto nulla di più.

Vediamo quali sono le interrogazioni da associare al `page-schema` `Dipendente`:

1. Per gli attributi: `Nome`, `Cognome`, `Stipendio`, `Qualifica`, `Nome_Sede` e `Citta'_Sede`:

   ```
   SELECT nome, cognome, stipendio, qualifica, sede, citta
   FROM Dipendente, Sede
   WHERE sede = nome_sede AND codice_dip = ?cod_dip?
   ```

2. Per l'attributo `Progetti`:

   ```
   SELECT nome_progetto, codice_pro
   FROM Partecipa, Progetto
   WHERE progetto = codice_pro AND dipendente = ?cod_dip?
   ```

Al fine di specificare in modo chiaro le interrogazioni associate ad un `page-schema` si propone infine la seguente sintassi:

```
DB to page-schema <nome_page_schema>
parameter (<nome_parametro_1>, ..., <nome_parametro_k>)
(
<nome_attributo_1>, ..., <nome_attributo_m>: <INTERROGAZIONE_SQL>;
....
<nome_attributo_p>, ..., <nome_attributo_n>: <INTERROGAZIONE_SQL>;
)
```

Ad esempio, per lo schema del dipendente fino ad ora usato nelle esemplificazioni, la specifica delle interrogazioni associate risulterebbe la seguente:

```
DB to page-schema Dipendente
parameter (cod_dip)
(
Nome, Cognome, Stipendio, Qualifica, Nome_Sede, Citta'_Sede:
   SELECT nome, cognome, stipendio, qualifica, sede, citta
   FROM Dipendente, Sede
   WHERE sede = nome_sede AND codice_dip = ?cod_dip? ;
Progetti:
   SELECT nome_progetto, codice_pro
   FROM Partecipa, Progetto
   WHERE progetto = codice_pro AND dipendente = ?cod_dip? ;
)
```

Esercizi proposti

1. Completare lo schema logico del sito dell'esempio **SITO AZIENDALE** con la specifica dei parametri e delle interrogazioni SQL da associare ad ogni `page-schema`.

2. Completare lo schema logico del sito del corso di formazione con la specifica dei parametri e delle interrogazioni SQL da associare ad ogni `page-schema`.

6 Una architettura software per la realizzazione di siti web centrati sui dati (Data-intesive Web Applications)

In questa sezione si presenta un'architettura software per la realizzazione di applicazioni web che generano un certo insieme di pagine web dinamiche. Nelle sezioni 2 e 3 si sono presentate le tecnologie Java Servlet e Java Server Pages rispettivamente e si sono mostrati esempi di pagine dinamiche generate usando in alternativa l'una o l'altra tecnologia. In un certo senso le soluzioni viste corrispondono alla realizzazione di software secondo l'approccio banale che prevede la stesura di un unico programma monolitico. Tale soluzione non è certamente quella suggerita dalle metodologie studiate dall'ingegneria del software che, al contrario, si basano sempre sulla suddivisione del software in moduli che interagiscono tra loro.

Anche per la realizzazione di un sito web centrato sui dati, e quindi per la realizzazione della corrispondente applicazione web, proponiamo di seguito un'architettura software modulare. Tale architettura si basa sul paradigma *Model View Controller* (MVC) nella versione 2.

Il paradigma MVC-2 prevede che la progettazione dell'applicazione venga suddivisa su tre livelli:

- *Application logic layer (Model)*: specifica la logica dell'applicazione, per quanto riguarda l'applicazione web che stiamo considerando la logica si traduce nell'estrazione dal database server dei dati e nella loro elaborazione;
- *Presentation layer (View)*: specifica le modalità e la forma della presentazione dei dati all'utente finale;
- *Control layer (Controller)*: specifica il flusso dell'applicazione e controlla l'interazione tra i due precedenti livelli.

Per avere una visione d'insieme della organizzazione in moduli che si propone, in Figura 9 si riporta lo schema dell'architettura dell'applicazione web. Si noti in particolare che:

- il livello Model viene realizzato da un unica classe Java ausiliaria, che chiamiamo *Dbms*, che interagisce con il database server; in tale classe è presente una costante stringa con la specifica SQL di ciascuna delle interrogazioni da sottoporre al database server, un metodo *extractXxxx* per ogni interrogazione da eseguire e un metodo *makeBeanXxxx* per ogni Java Bean da riempire con il risultato di un'interrogazione; tale livello viene completato da un certo numero di classi Java Bean, una per ogni tipo di risultato di interrogazione da gestire;
- il livello View viene invece implementato da un certo numero di Java Server Pages; tali JSP ricevono i Java Bean contenenti le informazioni da pubblicare nelle pagine e producono la pagina HTML da inviare al browser;
- infine, il livello Controller viene implementato in un unico modulo costituito dalla *Servlet Main*. Questa unica servlet gestisce tutte le richieste HTTP che giungono dalla rete e a seconda dei parametri presenti nella richiesta attiva metodi *extractXxxx* per l'estrazione di dati dal database server, e passa i Java Beans ottenuti alla JSP corretta per la presentazione in HTML.

L'intera architettura si basa sull'interazione tra Servlet e JSP e sull'uso di Java Bean per la rappresentazione del risultato di interrogazioni. A quanto già presentato in precedenza sulla tecnologia delle Java Servlet e delle JSP va aggiunta la presentazione dei meccanismi che consentono di: (i) trasferire Java Bean da una servlet a una JSP, (ii) attivare una JSP da una servlet e (iii) manipolare Java Bean dalle

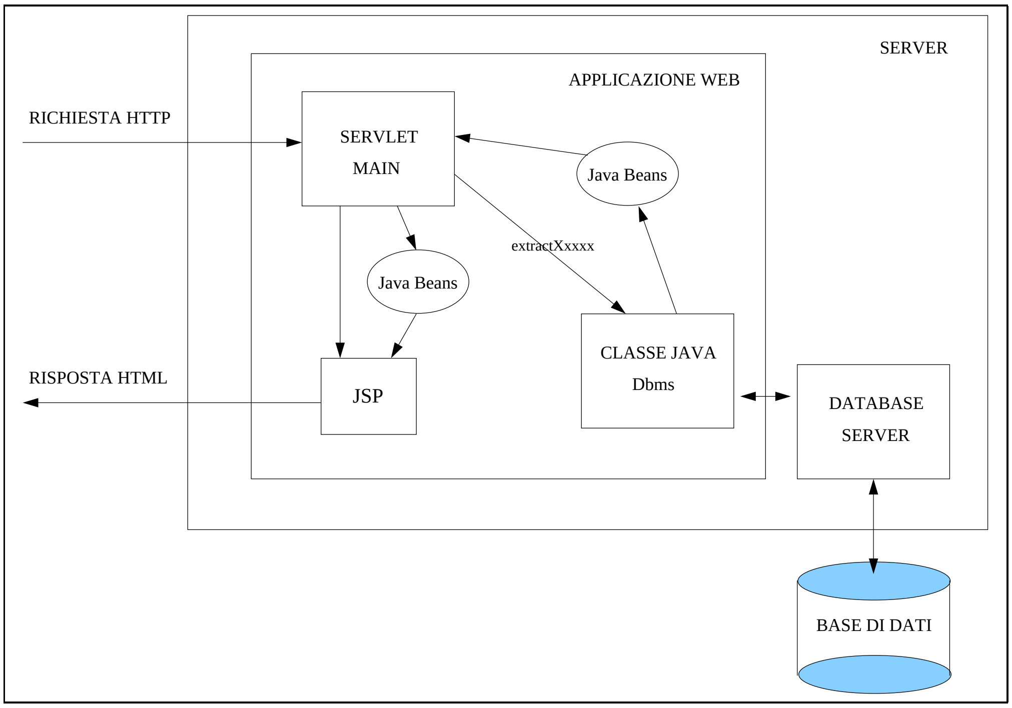

Figura 9: Architettura MVC-2 servlet-centric per applicazioni web centrate sui dati.

JSP. Quest'ultimo argomento sarà oggetto della prossima sottosezione, si presentano invece di seguito il meccanismo di interazione servlet/JSP.

Per trasferire un Java Bean da una servlet ad una JSP o ad un altra servlet è sufficiente inserire il Java Bean nell'oggetto `request` che viene passato alla JSP al momento dell'attivazione. Per fare ciò l'oggetto da passare `bean` viene inseriro nell'oggetto `request` come attributo attraverso la seguente istruzione:

```
request.setAttribute("NOME_ATTRIBUTO", bean);
```

L'attivazione di una JSP da una servlet avviene attraverso la creazione di un oggetto della classe `RequestDispatcher` come di seguito indicato (si supponga di voler passare il controllo alla JSP di nome `Nome_JSP`).

```
RequestDispatcher rd;
rd = getServletContext().getRequestDispatcher("Nome\_JSP");
rd.forward(request, response);
```

Si noti che l'istruzione `rd.forward(request, response);` è quella che trasferisce effettivamente il controllo alla JSP e che tale istruzione riceve come parametri, da trasferire alla JSP, l'oggetto `request` e l'oggetto `response`.

6.1 Marcatori speciali per la manipolazione di Java Bean da JSP

Visto l'uso frequente di oggetti di classi Java Bean nella specifica di JSP, sono stati introdotti marcatori speciali (tag) per la manipolazione di oggetti Java Bean nelle JSP in modo da evitare la scrittura di codice JAVA.

Di seguito si elencano i passi per l'uso di un oggetto Java Bean in una JSP indicando quali marcatori possono essere usati in sostituzione del codice JAVA nei vari casi:

- *Inclusione del file contenente la definizione della classe Java Bean che si intende utilizzare.* Tale operazione si ottiene attraverso la seguente direttiva `page`:

  ```
  <%@ page import="...BeanXxxx" %>
  ```

  che importa la definizione della classe `...BeanXxxx`.

- *Creazione di un oggetto Java Bean.* Tale operazione si ottiene attraverso l'uso del seguente marcatore speciale:

  ```
  <jsp:useBean id="ID_BEAN" class="NOME_CLASSE" scope="CONTEXT" />
  ```

  oppure

  ```
  <jsp:useBean id="ID_BEAN" class="NOME_CLASSE" scope="CONTEXT">
     CODICE DI INIZIALIZZAZIONE DEL BEAN
  </jsp:useBean>
  ```

  dove: l'attributo `id` indica il nome dell'oggetto creato, l'attributo `class` indica la classe Java Bean a cui l'oggetto creato appartiene e l'attributo `scope` indica il periodo di vita del bean; in particolare, i valori possibili per l'attributo `scope` sono:

  - `page` (valore di default): indica che la vita del bean corrisponde alla vita della pagina JSP dove viene creato: il bean non viene passato ad altre servlet o JSP eventualmente invocate dalla JSP corrente.
  - `request`: indica che la vita del bean corrisponde alla vita di tutti gli oggetti che vengono generati per la gestione di una richiesta, vale a dire la JSP corrente e anche tutte le eventuali servlet o JSP invocate da questa. Inoltre, il bean in questo caso non viene generato come nuovo oggetto, ma viene estratto dall'attributo di nome `ID_BEAN` dell'oggetto `request` passato alla JSP. Quindi, il valore `request` per l'attributo `scope` è il più usato nell'architettura software proposta, in quanto consente di manipolare gli eventuali oggetti Java Bean che vengono passati alla JSP dalla servlet che implementa il *Controller* dell'approccio MVC-2.
  - `session`: indica che la vita del bean corrisponde alla durata della sessione di connessione (richiede la gestione esplicita di sessioni di connessione)
  - `application` : indica che la vita del bean corrisponde alla durata dell'intera applicazione.

- *Accesso alle proprietà di un oggetto Java Bean.* Per accedere in lettura alle proprietà di un bean precedentemente generato è possibile usare il seguente marcatore speciale:

  ```
  <jsp:getProperty name="ID_BEAN" property="NOME_PROP" />
  ```

  Nel momento in cui viene generata la pagina corrispondente alla JSP, questo marcatore viene sostituito con il valore della proprietà `NOME_PROP` del bean `ID_BEAN`.

- *Modifica dlle proprietà di un oggetto Java Bean.* Per accedere in scrittura alle proprietà di un bean precedentemente generato è possibile usare il seguente marcatore speciale:

  ```
  <jsp:setProperty name="ID_BEAN" property="NOME_PROP" value="VAL" />
  ```

  Nel momento in cui viene generata la pagina corrispondente alla JSP, questo marcatore consente di assegnare il valore VAL alla proprietà NOME_PROP del bean ID_BEAN. Questo marcatore viene usato per l'eventuale inizializzazione di un bean nel contesto di un marcatore <jsp:useBean ...>INIZIALIZZAZIONE</jsp:useBean>.

Riferimenti bibliografici

[ACM01] D. Alur, J. Cruspi, and D. Malks. Core J2EE Patterns: Best Practices and Design Strategies. 2001, Prentice Hall.

[ACPT99] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone. Basi di dati (seconda edizione). 1999, McGraw-Hill Libri Italia srl.

[ACPT01] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone. Database Systems: Concepts, Languages and Architectures 2001, McGraw-Hill.

[FKB02] D. K. Fields, M. A. Kolb, S. Bayern. Web Development with JAVA SERVER PAGES. 2002, Manning.

[H01] M. Hall. core Servlets and Java Server Pages. 2001, Sun Microsystem Press Prentice Hall.

[M01] S. Mizzaro. Introduzione alla programmazione con il linguaggio Java. Franco Angeli, Milano, 2001.

[S01] W. Savitch. Java: An Introduction to Computer Science & Programming. Prentice Hall, 2001