Accesso a basi di dati via Web

Nicola Aloia, Cesare Concordia
CNUCE – Istituto del Consiglio Nazionale delle Ricerche
Via S. Maria 36, 56126 Pisa

E-mail N.Aloia@cnuce.cnr.it

Sommario. Nel presente lavoro sono analizzate le problematiche relative alla realizzazione di sistemi informativi basati sull'uso di sistemi di gestione di basi di dati tramite Internet. Sono descritte le tecnologie attualmente disponibili e le architetture di sistema realizzabili. In particolare, sono presentate le modalità d'accesso tramite lo schema HTTP/CGI, e attraverso applet Java che interfacciano il sistema di base di dati mediante le classi JDBC. Sono inoltre illustrate le modalità di realizzazione di applicazioni Client/Server con l'uso di Servlet Java e di applicazioni Object Oriented distribuite tramite il middleware CORBA e le classi RMI di Java. Infine è brevemente presentata la realizzazione di un sistema informativo basato sulla tecnologia qui illustrata.

Introduzione

All'inizio della sua storia il *web* si presenta come un sistema in cui le informazioni derivano da dati memorizzati staticamente su file, organizzati in file system gerarchici, la cui rappresentazione e diffusione avviene quasi esclusivamente tramite l'uso del formalismo HTML.

La vasta e rapida diffusione di Internet, ha portato negli ultimi anni un notevole incremento della complessità strutturale e comportamentale dei dati accessibili e distribuiti via Web, determinando un enorme arricchimento dell'informazione fruibile. Un grande impulso verso un sistema informativo *globale* è stato fornito principalmente da due fattori: l'integrazione sul Web di sistemi di gestione di basi di dati e l'utilizzo di "*browser intelligenti*", che consentono l'esecuzione locale di "*codice mobile*", cioè programmi trasferiti via rete durante una connessione al server. La caratteristica fondamentale di questi programmi dovrebbe essere l'indipendenza da qualsiasi piattaforma hardware/software ed una ragguardevole "*leggerezza*" nei consumi, per non causare significativi incrementi dei tempi di trasmissione o eccessivo overhead sul *client* in cui vengono trasferiti ed eseguiti. L'utilizzo combinato di questi due fattori, è oggi sempre più frequente nella realizzazione di siti Web ad elevato contenuto informativo; la maggior parte dei dati è fornita da un sistema di gestione di basi di dati e le "*pagine*" informative sono create dinamicamente mentre vengono inviate al browser. L'utilizzo di codice mobile consente di aumentare l'efficacia e la fruibilità delle informazioni nonché le prestazioni e la navigabilità dei siti stessi.

Molte soluzioni sono state adottate per l'utilizzo di basi di dati via Web, in gran parte dipendenti dal DBMS utilizzato o dal sistema operativo del server. Questo articolo illustra le caratteristiche fondamentali dell'offerta tecnologica attualmente disponibile per la realizzazione di applicazioni per basi di dati accessibili via Web, con particolare attenzione alle soluzioni che garantiscono indipendenza dal DBMS e dal sistema operativo in cui opera il server.

Lo studio e la sperimentazione di questi strumenti sono stati da noi applicati nella realizzazione di un sistema informativo sulla formazione ambientale, fruibile attraverso Internet (http://*www.anfora.cnuce.cnr.it*), per il ministero dell'Ambiente, in collaborazione con l'Istituto per lo Sviluppo della Formazione Professionale dei Lavoratori (ISFOL) [1].

1. Le modalità d'accesso ai dati

Il problema essenziale da risolvere per l'accesso a basi di dati via Web consiste nella realizzazione di una trasmissione, bidirezionale, rapida e corretta di dati tra browser e sistema di gestione di basi di dati. Schematizzando, il processo consiste nel trasformare la richiesta, che l'utente effettua mediante un browser, in una query che deve essere inviata al sistema di gestione di basi di dati per essere eseguita, il risultato dell'esecuzione della query deve essere inviato in risposta al browser per poter essere visualizzato. Nel seguito descriviamo le soluzioni attualmente disponibili, per la realizzazione di questo processo.

1.1 La soluzione HTML con schema HTTP/CGI.

La soluzione tipica adottata nella maggior parte delle attuali realizzazioni di siti Web, è quella che utilizza lo schema HTTP/Common Gateway Interface (HTTP/CGI) per la comunicazione client/server e la definizione dell'interfaccia utente tramite direttive HTML al browser (fig. 1). Questa soluzione comprende tre entità che interagiscono: *Browser*, *Web Server* e *DBMS*, questi ultimi possono risiedere (e molto spesso è così) su host diversi connessi via rete. Il browser riceve dal Web Server un documento HTML contenente specifiche di interfaccia utente per la formulazione di una richiesta al sistema di gestione di basi di dati, tipicamente con l'utilizzo di *form*. Nella definizione dell'interfaccia utente è dichiarato il nome del modulo (*CGI script*) da attivare sul server ("*action*") ed il metodo tramite il quale Web Server e script comunicheranno ("*method*"), che può essere GET o POST (nel primo caso i parametri sono passati tramite variabili di ambiente, nel secondo via stdin). I parametri forniti tramite l'interfaccia utente sono trasferiti via HTTP al Web Server, che attiva lo script CGI che a sua volta: legge e decodifica i parametri, apre una connessione con il DBMS (usando librerie native del sistema di gestione di basi di dati, specifiche per il sistema operativo in cui opera il server), esegue la query ed invia il risultato al Web Server che lo spedisce al browser sotto forma di un nuovo documento HTML.

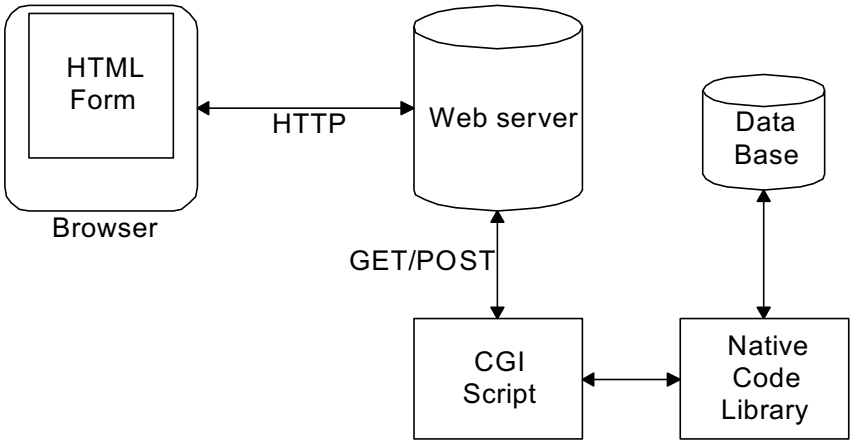

Fig. 1 Utilizzo di HTML con schema HTTP/CGI.

I limiti principali di questa soluzione sono:

1. L'interfaccia utente definibile mediante HTML è per sua natura statica, e quindi, in generale, poco efficace. Infatti, non è possibile controllare la bontà dei dati forniti dall'utente, o fornire a questi strumenti d'ausilio dipendenti dal contesto in cui opera, a meno di continue interazioni col server (soluzione non realistica, dati i tempi di risposta).

2. Il programma CGI deve essere scritto in un linguaggio che dipende dal sistema operativo nel quale risiede il Web Server ed inoltre per interagire con il DBMS, spesso, usa librerie esterne scritte in un linguaggio diverso. Questa soluzione ha il limite fondamentale della portabilità in ambienti differenti, e inoltre pone grossi problemi per la manutenzione e l'ottimizzazione del software.

3. Lo svolgimento di azioni cooperative tra client e server è impossibile, principalmente per il fatto che HTTP è un protocollo "*stateless*"; il che implica l'attivazione e l'esecuzione di un modulo CGI per ogni invocazione che arriva dai browser. Questa caratteristica può generare notevole overhead ed essere la fonte principale di degrado delle prestazioni o di blocchi del Web Server.

1.2 Le soluzioni dei produttori di software.

I principali produttori sia di Web Server che di sistemi di gestione di basi di dati forniscono degli strumenti per cercare di superare i limiti dello schema appena presentato.

Per gli aspetti legati al Server hanno realizzato ed integrato nei loro prodotti, delle librerie di funzioni specifiche per la realizzazione di CGI script. Tra i più noti produttori citiamo Netscape che ha integrato nei propri Server due prodotti: Web

Application Interface (WAI) e Netscape Server API (NSAPI), Microsoft che distribuisce Internet Server API (ISAPI), Oracle che fornisce, insieme ai propri sistemi per basi di dati, WebServer API. Gli script creati utilizzando queste librerie sono a volte chiamati *fast CGI script.* L'obiettivo è duplice: ridurre l'overhead derivante dall'esecuzione dei CGI e supplire alla mancanza di stati del protocollo HTTP. Le librerie forniscono funzionalità di accesso ai database sia tramite ODBC (WAI, NSAPI e ISAPI) che, come nel caso del Web Server API, l'accesso diretto (ma solo ai database Oracle).
Dal lato client le principali soluzioni proposte sono di tre tipi:

1. l'uso di *plug in*, cioè particolari programmi che devono essere installati sul client, e che implementano la connessione con il sistema di gestione di basi di dati,
2. l'uso di script *embedded* nel documento HTML e, a volte, l'estensione del formalismo HTML stesso tramite l'introduzione di *tag* non standard,
3. l'uso di codice mobile proprietario (Omniware , Safe TCL)

Il limite principale delle soluzioni proprietarie sul Web, deriva dal non rispetto dell'indipendenza hardware/software. Esse infatti non solo sono vincolate al Web Server ed al sistema operativo del server, ma possono esserlo anche al browser (è il caso delle estensioni di HTML) o al sistema operativo del client .

1.3 La soluzione Java.

Java è un linguaggio di programmazione Object Oriented, multithreaded e multipiattaforma [4], [3]. I programmi sorgenti Java una volta compilati sono trasformati in un formato, denominato *bytecode*, che è interpretato dalla macchina virtuale Java, disponibile nella maggior parte dei sistemi operativi. In particolare i principali browser oggi in circolazione hanno tutti integrata una Java VM ("*Java enabled browsers*"). Oltre a realizzare applicazioni complete in un linguaggio di programmazione ad oggetti, il successo di Java deriva dalla sua integrazione con i browser; infatti è possibile inserire in documenti scritti in HTML dei link a programmi Java , i quali vengono trasferiti sul computer dove risiede il browser ed eseguiti in fase di visualizzazione del documento HTML (fig. 2).

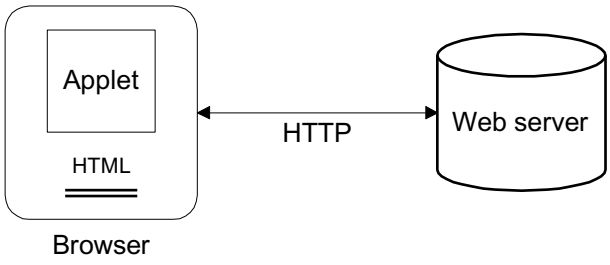

Fig. 2 Utilizzo di Java via HTML..

Questi programmi prendono il nome di "*applet*". Per la loro natura, di programmi trasferiti via rete ed eseguiti localmente, le funzionalità degli applet sono sottoposte a notevoli vincoli di sicurezza. Gli applet non possono attivare un programma locale o scrivere/leggere il contenuto di file locali, inoltre non possono comunicare con altri host che non siano il server dal quale sono stati trasferiti. A parte questi limiti comunque un applet è un programma Java a tutti gli effetti. E' possibile ad esempio creare una interfaccia *user friendly* per l'immissione dati, utilizzando oggetti grafici, inserire routine di controllo etc.

L'accesso a basi di dati tramite il linguaggio Java avviene con l'uso di una libreria di classi, distribuite gratuitamente con l'ambiente di sviluppo, note come JDBC. Nel prossimo capitolo descriveremo in maniera più dettagliata le funzionalità delle classi JDBC e le architetture realizzabili con questa soluzione.

2. JDBC

Le Java Data Base Connectivity (JDBC) sono classi, scritte in linguaggio Java, che consentono di interfacciare applicazioni Java con sistemi di gestione di basi di dati [5]. Le classi JDBC sono progettate, com'è nella filosofia Java, per essere *"platform indipendent*". Per garantire l'indipendenza dai sistemi di gestione di basi di dati, le classi JDBC utilizzano dei *driver manager*, essi si occupano di gestire la connessione fra le applicazioni Java e gli specifici *driver*. I driver sono configurabili dall'ambiente del sistema operativo e costituiscono le varie implementazioni delle classi JDBC. Con questa architettura (fig. 3) si garantisce l'indipendenza dell'applicazione Java da qualsiasi DBMS, infatti per utilizzare uno specifico prodotto per basi di dati, basterà configurare l'opportuno driver senza intervenire nel codice applicativo.

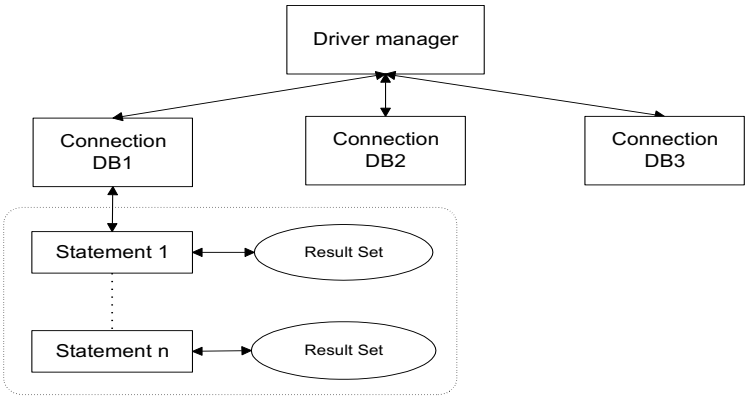

Fig. 3 Schema di accesso ai dati via JDBC.

Le classi principali che costituiscono JDBC sono:

1. *java.sql.DriverManager*, che permette di caricare i drivers ed effettuare le connessioni con i sistemi di gestione di basi di dati.
2. *java.sql.Connection*, che gestisce la connessione ad un particolare database.
3. *java.sql.Statement*, che permette l'esecuzione di istruzioni SQL in una determinata connessione.
4. *java.sql.ResultSet*, che permette di gestire il set di risultati prodotti dallo statement SQL.
5. *java.sql.DatabaseMetaData*, che consente di interrogare il catalogo del sistema di gestione di basi di dati.

Esistono quattro tipi di driver JDBC:

1. *ODBC-JDBC drivers*: questi driver convertono le chiamate JDBC in chiamate alle API ODBC.
2. *native-API partly-Java driver:* convertono le chiamate JDBC in chiamate alle API native del particolare sistema di gestione di basi di dati
3. *net-protocol all-Java driver:* le chiamate JDBC vengono inviate ad un server di rete tramite un protocollo di comunicazione, è il server che provvede a convertirle in chiamate native del sistema di gestione di basi di dati e viceversa.
4. *native-protocol all-Java driver:* questi driver consentono di effettuare le chiamate JDBC direttamente nel protocollo di rete del sistema di gestione di basi di dati, in questo modo l'interazione tra applicazione Java e DBMS è diretta.

I driver di tipo 1 sono distribuiti dalla JavaSoft, lo scopo è quello di sfruttare la diffusione di ODBC; molte case produttrici di software hanno rilasciato JDBC di tipo 2 specifici per i sistemi di gestione di basi di dati più diffusi. Il grosso limite dei primi due tipi di driver, nella realizzazione di applicazioni client/server, deriva dalla necessità di installare librerie di software (ODBC o native dello specifico DBMS) sul client. Le potenzialità delle JDBC nello sviluppo di applicazioni client/server sono espresse maggiormente nei driver di tipo 3 o 4, con essi è infatti possibile stabilire una connessione mediata (3) o diretta (4) tra client e sistema di gestione di basi di dati, e ciò vale anche nel caso di applicazioni client/server sul Web.

2.1 Architettura dei sistemi WEB che usano JDBC

Ciascuna delle implementazioni dei driver JDBC, descritte nel paragrafo precedente, permette di realizzare applicazioni client/server, per l'accesso a basi di dati, la cui architettura può essere ad uno o più livelli. Per livelli di un'architettura intendiamo i diversi strati di software tra l'applicazione ed il sistema di gestione di basi di dati. Descriviamo di seguito le architetture che si possono realizzare utilizzando i vari driver JDBC.

Sistemi con architettura ad un livello L'applet contiene oltre al codice per la realizzazione dell'interfaccia utente, anche il codice relativo al Driver Manager e quello relativo alle classi JDBC. In questo modo, dal browser è possibile interrogare direttamente il database, purché esso sia presente nello stesso host da cui è stato trasferito l'applet (vincolo di sicurezza degli applet). Una tale soluzione può essere adottata solo se si hanno a disposizione driver JDBC di tipo 4.

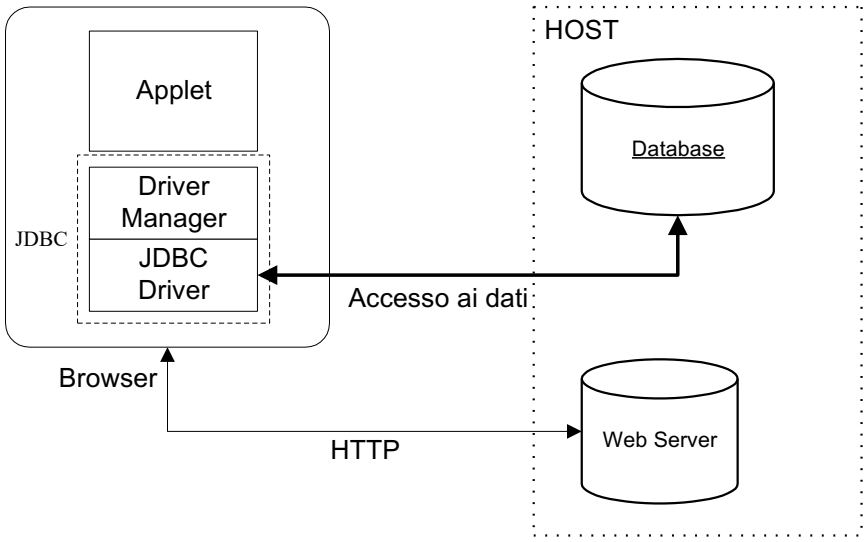

Fig. 4 Sistemi con architettura ad un livello.

Sistema con architettura a due livelli. Utilizzando driver JDBC di tipo 1 o 2 è possibile realizzare sistemi con una architettura a due livelli. Al primo livello abbiamo l'applet che contiene il codice dell'interfaccia utente, il driver manager ed il driver JDBC, al secondo livello troviamo le API native del particolare sistema di gestione di basi di dati, specifiche per il sistema operativo su cui è installato. E' necessario che l'applet interagisca con le API per poter accedere al database.

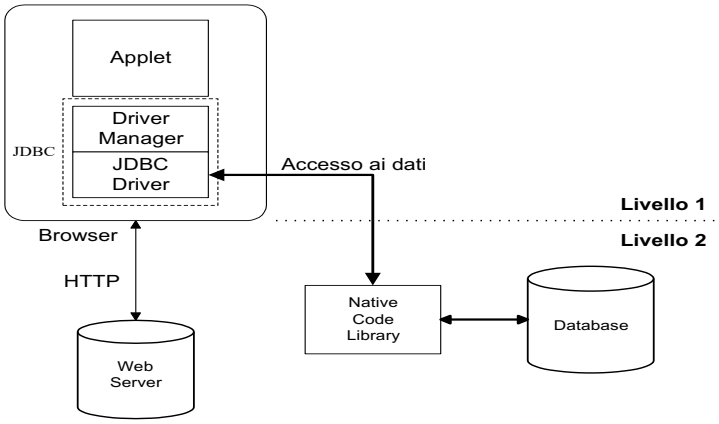

Fig. 5 Sistemi con architettura a due livelli.

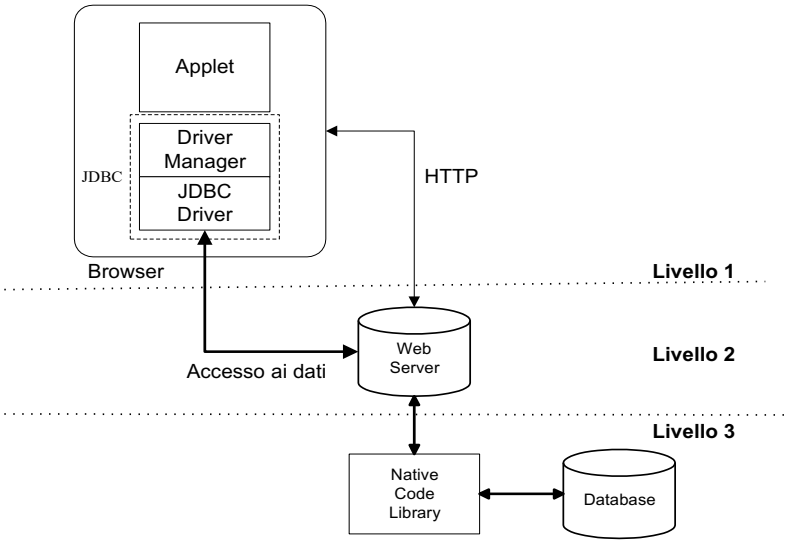

Fig. 6 Sistemi con architettura a tre livelli.

Sistemi con architettura a tre livelli. Utilizzando driver JDBC di tipo 3 è possibile realizzare sistemi con una architettura a tre livelli come quella in fig. 6. Questa soluzione richiede l'utilizzo di Web server specifici per i driver JDBC, i quali hanno

un'architettura client/server, la cui componente server e realizzata dal Web server. Anche in questo caso l'applet (primo livello) comprende il codice per la realizzazione dell'interfaccia utente e per il Driver Manager, inoltre contiene la componente client del driver JDBC. Il web server (secondo livello) interagisce con l'applet e con le API native del DBMS (terzo livello), che fungono da gateway per l'accesso al sistema di gestione di basi di dati. Questa soluzione non richiede che il database risieda sullo stesso host.

Analisi dei modelli. Per l'accesso ad un sistema di gestione di basi di dati via Web, le soluzioni che si possono realizzare, utilizzando i driver JDBC, descritti nel precedente paragrafo, sono le due seguenti:

- Sistemi con architettura ad un livello
- Sistemi con architettura a tre livelli

Il sistema con architettura a due livelli non è applicabile sul web in quanto richiede la presenza sul client delle componenti software di entrambi i livelli, cioè dei driver ODBC o delle librerie di API native per lo specifico DBMS (driver JDBC di tipo 1 e 2 rispettivamente). In certe condizioni è possibile realizzare tale architettura di sistema nell'ambito di una intranet.

La realizzazione effettiva di sistemi con architettura ad uno o a tre livelli, per l'accesso a basi di dati via Web, presenta, allo stato attuale della tecnologia, particolari inconvenienti pratici. Nel caso di architettura ad un livello infatti è necessario dapprima stabilire una connessione HTTP per trasferire l'applet, quindi una seconda connessione su una diversa porta IP per l'accesso al database, questo comportamento potrebbe essere ritenuto non lecito da un "*firewall*" che di conseguenza impedirebbe la connessione al database. Un'architettura di sistema di questo tipo è quindi attualmente più adatta per realizzazioni di sistemi intranet. In architetture di sistema a tre livelli, il Web Server deve svolgere la doppia funzione di trasferimento dei dati verso il browser e di gestore delle connessioni con il database. Il limite di questa soluzione deriva dal fatto che non tutti i Web Server attualmente in circolazione hanno questa capacità, perciò per realizzare questa architettura è necessario installare sul sito del software ad hoc. In entrambi i casi, inoltre, l'applet deve occuparsi di gestire i flussi dei dati in arrivo dal database e la loro visualizzazione, questo comporta un aumento della complessità del codice e quindi di un maggior overhead in fase di esecuzione sul client. Esiste una terza soluzione che unisce i vantaggi derivanti dall'uso del linguaggio Java (applet, multithread, JDBC) con la semplicità di realizzazione, questo modello si basa sull'uso dei "*servlet*" Java, che descriviamo nel prossimo paragrafo.

3. Servlet

Abbiamo visto, nei paragrafi precedenti, che in applicazioni client/server, l'utilizzo degli applet Java consente di "*estendere*" dinamicamente le funzionalità della parte client dell'applicazione. Allo stesso modo è possibile creare programmi

Java chiamati "*servlet*" [2] che permettono di estendere le funzionalità della parte server dell'applicazione. I *servlet* e gli *applet* presentano numerose analogie:

- L'utilizzo di servlet richiede l'esistenza nel server su cui saranno eseguiti di una macchina virtuale Java,
- I servlet possono essere memorizzati su host diversi dal sito a cui fa riferimento l'applet; analogamente agli applet, essi saranno trasferiti sul server quando il client ne richiederà l'esecuzione.

A differenza degli applet però i servlet sono *faceless object* ossia non forniscono la possibilità di implementare interfacce utenti, essi interagiscono solo con il server di rete. I servlet sono indipendenti dal protocollo di rete usato, le classi Java Servlet forniscono dei metodi nativi per interagire con un generico server di rete indipendentemente dal servizio che esso fornisce o dal sistema in cui opera.

I principali vantaggi dell'uso dei servlet rispetto ai normali CGI script sono:

- Sono scritti in Java, che è un linguaggio multi-piattaforma; è possibile quindi scrivere servlet che possono essere eseguiti sia in remoto che localmente da più Web Server indipendentemente dal sistema operativo su cui sono stati implementati.
- L'interazione tra servlet e server di rete avviene tramite i metodi forniti dalle classi Java Servlet, questo garantisce indipendenza dal generico Server e dai protocolli di comunicazione e quindi la possibilità di riuso.
- Le classi Java Servlet consentono di scrivere programmi rientranti. Il servlet viene caricato una volta per tutte in memoria dalla VM Java la prima volta che viene invocato, e continua a risiedervi sino a quando non viene scaricato esplicitamente; questa caratteristica consente di migliorare notevolmente le prestazioni del sito rispetto all'uso di script CGI, inoltre permette la condivisione di informazioni tra connessioni diverse e la riduzione del numero di processi attivi sul Server.

La struttura tipica dei servlet è composta da tre parti:

1. *Init* questa parte comprende tutte le operazioni che sono eseguite solamente la prima volta che il server attiva il servlet caricandolo in memoria
2. *Core* le operazioni che il servlet esegue ogni volta che viene richiamato.
3. *Destroy* le operazioni che sono compiute solo quando il servlet viene scaricato dalla memoria

3.1 Sistemi con architettura a tre livelli con uso dei servlet Java.

La realizzazione di un sistema con architettura a tre livelli usando i *servlet* non richiede l'installazione di specifici Web Server, essendo disponibili librerie di classi per la realizzazione di *servlet*, per i principali server di rete. Grazie a queste classi ed

alle JDBC è possibile realizzare programmi Java che implementano completamente le funzionalità di *gateway* tra un Web Server ed un sistema di gestione di basi di dati (fig. 7).

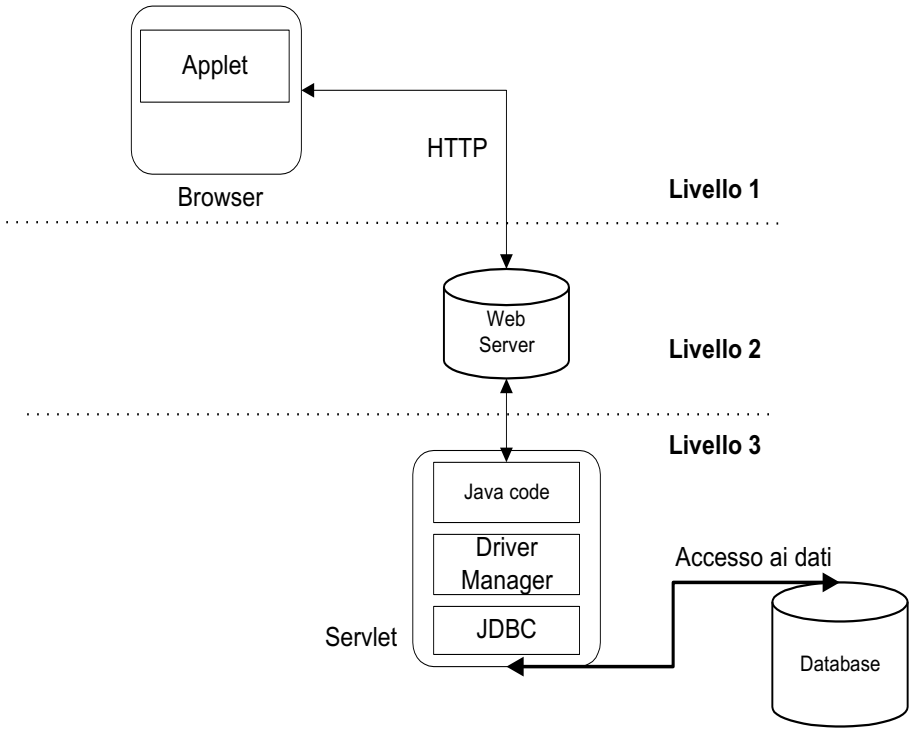

Fig. 7 Architettura a tre livelli con servlet.

Il primo livello è costituito da un applet che costituisce l'interfaccia con cui l'utente interagisce con la base di dati. Quando una richiesta viene inviata al Web Server, questi localizza il *servlet* (eventualmente trasferendolo via rete) e lo attiva passandogli i parametri. Il *servlet* interagisce con il database e fornisce i risultati al Web Server che li restituisce al client.

4. Evoluzioni: *Object Web*

Java introduce un nuovo modello di applicazioni client/server sul Web fornendo la possibilità di creare codice mobile (gli applet) con il quale realizzare componenti client multi-piattaforma. L'introduzione degli applet, nelle applicazioni distribuite sul

Web, è considerata da molti il primo passo verso la creazione dell'*Object Web* [6], ossia la possibilità di realizzare applicazioni Object Oriented distribuite. Questo richiede necessariamente la realizzazione di una infrastruttura di comunicazione che consenta l'interazione di oggetti distribuiti, ossia oggetti che possono essere localizzati su host acceduti via rete. I principali candidati a costituire tale infrastruttura sono due: *Common Object Request Broker Architecture* (CORBA) e *Distributed Component Object Model* (DCOM). CORBA è il risultato del lavoro di *Object Management Group* (OMG), un consorzio di cui fanno parte le principali aziende produttrici di software eccetto Microsoft che propone DCOM. Tramite tali infrastrutture è possibile realizzare applicazioni object oriented distribuite sul Web, inoltre non è necessario che gli oggetti che interagiscono siano scritti nello stesso linguaggio. Le differenze principali fra CORBA e DCOM sono legate soprattutto al tipo di approccio che ha portato alla loro realizzazione: CORBA è un sistema aperto, completamente Object Oriented, DCOM è legato all'implementazione in ambienti Microsoft, anche se la sua integrazione in Visual J++ (ambiente di sviluppo Java della Microsoft) lascia supporre l'intenzione, da parte del produttore, di estenderne le funzionalità ad ambienti multipiattaforma.

OMG definisce le specifiche di CORBA in un linguaggio neutro che prende il nome di *Interface Definition Language (IDL)*. Tramite IDL si definiscono le interfacce dei componenti la cui implementazione effettiva potrà avvenire in qualsiasi linguaggio, si crea cosi un *middleware* che consente ad oggetti scritti in linguaggi differenti di *interoperare*. L'attuale implementazione di Java consente (con le usuali restrizioni dovute alla sicurezza), ad un applet, di invocare un metodo di un oggetto Java residente su un host remoto tramite Remote Method Invocation (RMI). A differenza di CORBA, RMI non consente di effettuare la invocazione dinamica di metodi o la scoperta dinamica (*dynamic discovery*) di nuovi oggetti, inoltre RMI consente l'interazione solo di oggetti scritti in linguaggio Java. Le caratteristiche di Java (multipiattaforma, mutithread, codice mobile, free software) sembrano essere particolarmente adatte all'implementazione delle specifiche di CORBA, questo fa si che la tendenza attuale sia quella della piena integrazione tra Java e CORBA.

Lo schema di fig. 8 illustra la struttura possibile di applicazioni client/server via Object Web usando CORBA.

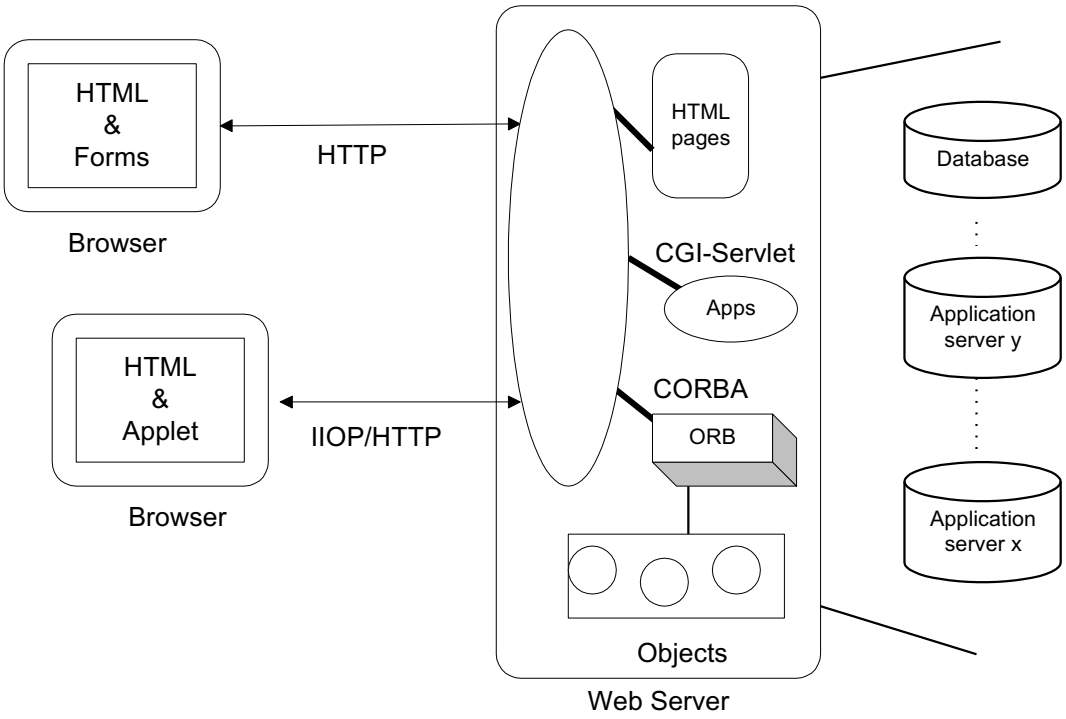

Fig. 8 Schema di applicazioni Java/Corba

La parte client può essere costituita da un normale documento HTML oppure da un applet, la comunicazione client/server può avvenire tramite il protocollo HTTP oppure tramite Internet Inter-ORB Protocol (IIOP), il protocollo di comunicazione usato da CORBA e basato su TCP/IP. Tramite IIOP è possibile per il client l'invocazione diretta di oggetti remoti. Gli oggetti interagiscono tra loro e con il server tramite il bus CORBA Object Request Broker (ORB) il quale provvede a gestire le invocazioni di metodi rendendo trasparente la localizzazione degli oggetti stessi. Nell'Object Web, CORBA costituisce il *middleware* per la comunicazione tra oggetti remoti, ma poiché anche IIOP usa Internet come backbone è possibile scrivere applicazioni che utilizzano il tradizionale schema HTTP/CGI.

5. Applicazione ad un caso concreto

L'occasione per poter sperimentare l'uso della tecnologia appena descritta si è presentata nell'ambito del progetto "*Realizzazione e distribuzione di un sistema informativo sulla formazione ambientale*", oggetto di una convenzione tra il CNUCE e l'Istituto per lo Sviluppo della Formazione Professionale dei Lavoratori (ISFOL), che ha predisposto la ricerca su richiesta del Ministero dell'Ambiente, e avente lo scopo di rendere fruibile attraverso Internet il risultato di un censimento a livello nazionale delle attività formative inerenti tematiche ambientali svolte presso università, istituti di istruzione secondaria e centri di formazione professionale.

L'obiettivo che ci siamo posti, dal punto di vista informatico, è stato la realizzazione di un sistema che fosse indipendente da specifiche caratteristiche hardware e software delle piattaforme preposte alla sua gestione, così da lasciare la possibilità al committente di far migrare l'applicazione nell'ambiente che più ritenesse opportuno, senza ulteriori costi di sviluppo. Nel seguito commentiamo brevemente la soluzione adottata nella realizzazione della parte del progetto riguardante l'interrogazione della base di dati attraverso il Web.

L'architettura del sistema realizzato è a tre livelli. Le interfacce per l'interrogazione sono costituite da applet Java che permettono all'utente di specificare opportuni filtri di selezione sui dati. Ciascun applet è riferito ad una determinata tipologia dei dati presenti nel database e implementa la logica necessaria ad impedire, errori di digitazione ed a fornire ausili per la formulazione della richiesta. In fase di inzializzazione sul browser l'applet esegue delle query per richiedere al database i dati da proporre come scelte possibili nella presentazione dei filtri di selezione. L'interazione con il sistema di gestione di basi di dati avviene sul server per mezzo di servlet. Quando dal browser viene richiesta l'esecuzione di una query, il Web Server attiva il servlet opportuno, quest'ultimo costruisce la query e la esegue usando le JDBC, quindi passa il risultato al Web Server il quale lo rinvia di nuovo al client. Vi sono due tipi di query che danno vita a due comportamenti diversi da parte dei programmi:

- Alla richiesta dei valori di inizializzazione dei controlli costituenti l'interfaccia utente, i servlet rispondono inviando dati opportunamente codificati (per minimizzare i tempi di trasmissione), l'applet che è in *stand by* provvede alla decodifica e alla loro visualizzazione sul browser

- Alla richiesta di esecuzione di una query sul contenuto della base di dati, i servlet rispondono con una o più pagine HTML generate *on the fly* contenenti il risultato che viene visualizzato dal browser.

Il tempo di connessione al database comporta un overhead che paragonato ai tempi di risposta sul Web non è sicuramente un fattore critico. Per migliorare le prestazioni sul lato server, abbiamo adottato una soluzione consistente

nell'attivazione di un pool di connessioni ciascuna delle quali è gestita da un thread (connection broker) (fig. 9).

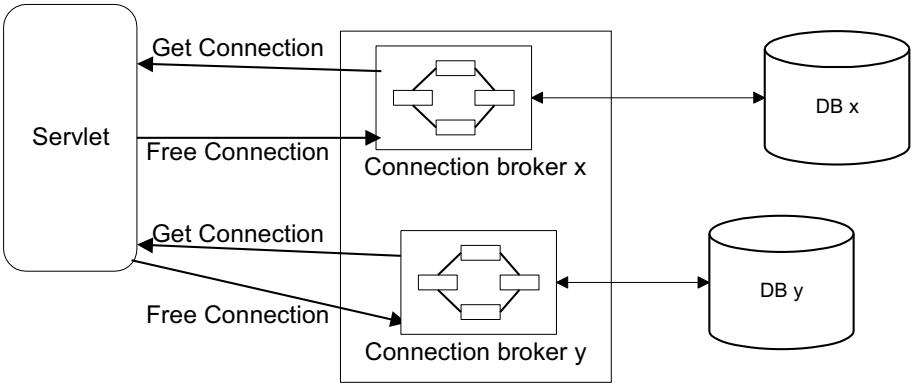

Fig. 9. Architettura del connection broker

Nella fase di *Init* il servlet attiva uno o più connection broker, ciascun broker apre un certo numero di connessioni con la base di dati e resta attivo in bakground. Quando il servlet nella fase *Core* deve eseguire una query, chiede al broker una connessione, terminata le operazioni la "restituisce" rendendola di nuovo disponibile. Per la caratteristica di "*persistenza*" in memoria dei servlet, il connection broker sarà condiviso dalle attivazioni successive del *core*. I broker saranno distrutti all'invocazione della fase *destroy* del servlet. Rispetto alle tradizionali implementazioni tramite CGI, si sfrutta la possibilità di condividere strutture dati dei servlet per connessioni successive ed a ridurre il numero di processi attivi sul server.

Abbiamo sperimentato, con successo, il sistema con due diversi DBMS (Access ed SQL Server della Microsoft) e due diversi *driver* senza apportare alcuna modifica al codice sorgente.

6. Conclusioni.

Nel presente lavoro, dopo una rassegna della tecnologia disponibile per la realizzazione di sistemi informatici via Web, che soddisfano il principio di indipendenza dalle varie piattaforme hardware/software, abbiamo analizzato le varie architetture realizzabili con gli strumenti attualmente disponibili ed abbiamo brevemente illustrato la soluzione adottata nella realizzazione di un sistema informativo.

Bibliografia

[1] N. Aloia, D. Canino, C. Concordia: “Un sistema informativo sulla formazione Ambientale” Rapporto interno CNUCE – Marzo 1998.

[2] The Java Servlet API, disponibile in versione elettronica all’indirizzo: http://java.sun.com/features/1997/aug/jws1.html.

[3] Gary Cornell, Cay S. Horstmann, “Core Java”, SunSoft Press

[4] Bruce Eckel “Thinking in Java”, Prentice Hall Computer Books, disponibile in versione elettronica all’indirizzo: http://www.EckelObjects.com/javabook.html

[5] Graham Hamilton, R. G. G. Cattell, Maydene Fisher “JDBC Database Access with Java A Tutorial and Annotated Reference”, Addison Wesley.

[6] Robert Orfali, Dan Harkey “Client/Server Programming with Java and CORBA” Wiley Computer Publishing