UNIVERSITÀ DEGLI STUDI DI CAMERINO

SCUOLA DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica

STUDIO SULLA MIGRAZIONE DI UNA APPLICAZIONE WEB, PER LA GESTIONE DI SOCIETÀ DI CALCIO DEL SETTORE GIOVANILE, IN J2EE

Tesi di Laurea

Laureando

Giovanni Sabatini

Relatore

Prof. **Fausto Marcantoni**

Correlatore

DOTT. NICOLA CORSARO

ANNO ACCADEMICO 2009 / 2010

Ringraziamenti

Si ringraziano per la disponibilità dimostrata il Dipartimento di Matematica e Informatica e in particolare Prof. Fausto Marcantoni.

Si ringrazia la società Xteam s.r.l e in particolare il Dott. Nicola Corsaro per il supporto tecnico, la competenza e collaborazione dimostrate.

Introduzione

In questo lavoro di tesi è stata studiata la migrazione di un applicativo scritto in PHP in J2EE. Si è trattato quindi di approfondire alcune caratteristiche dello standard J2EE, scegliere l'Application Server conforme alle specifiche e valutare gli aspetti positivi e negativi di tale lavoro.

Nel primo capitolo vengono descritte le caratteristiche generali dell'applicativo, con un breve cenno alla sua evoluzione e le motivazioni che hanno spinto all'impiego della tecnologia J2EE. Non vengono descritte nel dettaglio le funzionalità del programma.

Nel secondo capitolo sono trattate le proprietà principali di J2EE come il concetto di Business Logic, di persistenza dei dati, della separazione logica in livelli applicativi delle varie componenti di un programma e della sicurezza. In particolare sono descritte le caratteristiche principali di alcuni Enteprise Java Bean, degli Entity Bean, del concetto di servlet e servlet container.

Il terzo capitolo descrive il lavoro pratico di migrazione del programma con alcuni esempi pratici delle problematiche incontrate sia per quanto riguarda la configurazione dell'application server che dell'implementazione del codice.

Si è partiti da un problema pratico, sono stati approfonditi alcuni aspetti teorici ed infine è stata prodotta una soluzione che si è rilevata estremamente interessante sia dal punto di vista tecnologico che professionale.

Questo studio è stato svolto in collaborazione con la società Xteam di Perugia (http://www.xteam.it) che realizza applicativi gestionali aziendali di elevata complessità, con soluzioni sviluppate secondo lo standard J2EE. Il contesto di questo lavoro è diverso dai prodotti che Xteam sviluppa, soprattutto in riferimento al tipo di utenza a cui è rivolto il programma.
Durante lo svolgimento dello studio, è stato comunque molto utile potermi confrontare su alcune problematiche con i tecnici dell'azienda, con i quali è stato instaurato un proficuo e positivo scambio di opinioni.

Capitolo 1 - Programma di gestione sistema “Calcio”

1.1 Introduzione

La realizzazione di un applicativo, oggi, non può prescindere dalla considerazione che solitamente l’utenza fa ormai largo uso di programmi per l’accesso alla rete Internet e che, pertanto, anche programmi di carattere generale rientrano in questa ottica. Tutto questo, per le note limitazioni del protocollo http, ha causato lo sviluppo di un susseguirsi di strumenti e sistemi per la progettazione di software utilizzabile via Internet. I browser di qualche anno fa si sono trasformati sempre di più in pacchetti software che al loro interno contengono implementazioni che vanno oltre al paradigma relativo all’accesso e alla classica visualizzazione delle pagine html per cui erano nati originariamente. Il protocollo http ha forti limitazioni, quali la mancanza di gestione degli stati (stateless) di una connessione. E’ quindi stato necessario implementare sia lato server che client meccanismi per poter utilizzare questo protocollo anche nei casi in cui sia indispensabile conservare lo stato/i di una connessione http. Non va poi dimenticato che, per poter realizzare sistemi di visualizzazione dati, articolati, gradevoli ed efficienti, ai tradizionali browser, è stato integrato un linguaggio che permettesse tutto ciò. Questa integrazione, soprattutto nei primi anni 2000, ha causato vari problemi di compatibilità. In quel periodo infatti la guerra dei browser, nata alla fine degli anni ’90, aveva portato alla rincorsa dei vari vendor alla continua ricerca di nuove funzionalità e caratterizzazioni di javascript, fino a determinare seri problemi di incompatibilità di funzionamento per una stessa applicazione da un browser ad un altro. La fine di questa competizione ha portato alla cessazione del continuo sviluppo di nuove specifiche di javascript con il risultato di una riscoperta di questo linguaggio e del suo massiccio utilizzo per lo sviluppo di applicazioni “web”. Non a caso si parla di applicazioni e

non di pagine, concetto ormai superato o comunque fortemente riduttivo rispetto agli scenari attuali. La continua richiesta, da parte del management aziendale di applicazioni utilizzabili via Internet, si sposa con il valore strategico che questa tipologia di programmi, per la loro struttura, offre a livello commerciale. Avere la possibilità di poter gestire in modo centralizzato sia i dati, sia la logica del programma, la sua manutenzione, l’accounting e la gestione dei costi e la tipologia di licenze , impone sempre di più l’adozione di questo paradigma. Un’attenzione particolare va posta riguardo alla manutenzione e fruibilità dell’applicativo. Nel primo caso, la gestione della manutenzione non è distribuita sul territorio in funzione dei clienti che lo utilizzano, nel secondo è che il potenziale bacino d’utenza ed i conseguenti ricavi economici sono estremamente più elevati rispetto ai sistemi software sviluppati in modo tradizionale. In buona sostanza è stato abbattuto il muro fra la collocazione logistico/fisica del cliente con l’applicativo. Se il potenziale cliente ha una connessione ad Internet sarà in grado di utilizzare il software. In funzione quindi del potenziale bacino di utenza e della complessità che un applicativo potrebbe avere, va valutato l’utilizzo di strumenti di progettazione e sviluppo, di carattere Enterprise. L’esperienza di questo lavoro di tesi riguarda proprio questi aspetti, focalizzando l’attenzione alla migrazione di un esistente programma gestionale realizzato in php verso J2EE.

Il programma da me sviluppato per la gestione delle società di calcio del settore giovanile ha subito tre step di realizzazione. La prima versione fu realizzata con strumenti estremamente limitati come MS-ACCESS; è stata poi effettuata una prima implementazione web in PHP e in questo lavoro di tesi è stato studiato e realizzato il porting in J2EE.
Questo triplice passaggio ha permesso al sottoscritto, pur provenendo già da una esperienza sistemisitca, di valutare i pro e i contro delle varie tecnologie utilizzate e, soprattutto, pur se già conoscitore di alcuni linguaggi di programmazione, primo su tutti, lo “C”, gli ha permesso di

approfondire alcuni aspetti estremamente interessanti, sia dal punto di vista tecnico che strategico. Senza soffermarci sulla prima implementazione, nei paragrafi successivi, saranno trattati gli aspetti fondamentali della seconda e terza implementazione.

1.2 Considerazioni di carattere generale

La realizzazione di un programma gestionale, fruibile tramite i consueti strumenti utilizzati per l’accesso alla navigazione web, rispetto ai software tradizionalmente progettati senza l’utilizzo del protocollo http, da un lato implica una complessità maggiore, dall’altro da la possibilità di poter usufruire di informazioni, tecnologie e potenzialità superiori a quelle disponibili nella programmazione classica.
Si ha cioè a disposizione un maggior potenziale di integrazione tipica della comunità che utilizza la rete e quindi, rispetto alle classiche funzionalità di gestione di un programma, si possono aggiungere gli strumenti atti a migliorare lo scambio di informazioni fra utenti, possibilità di poter creare e gestire comunità, con il vantaggio di realizzare un prodotto più completo.

In linea generale il programma per la gestione del settore calcio giovanile, di seguito denominato programma, doveva rispettare le seguenti caratteristiche:

1. **multiutenza.** Gestire più utenze per una stessa società;
2. **multisocietà.** Gestire più società sportive;
3. **accessibilità**. Il programma doveva essere accessibile via Internet con l’accesso all’applicativo tramite un browser;
4. **facilità di utilizzo**. L’interfaccia del programma doveva risultare estremamente semplice ed intuitiva per l’utente;
5. **risorse di tipo open source.** sviluppo del programma con strumenti di tipo open-source;

6. **bacino d’utenza**. Il bacino d’utenza, che potenzialmente questa applicazione potrebbe avere, è dell’ordine delle decine di migliaia .

Con l’implementazione del programma in PHP erano state colmate alcune lacune rispetto alla prima versione (MS-ACCESS), erano stati eliminati i problemi di compatibilità fra le varie versioni di MS-ACCESS, il costo della licenza del pacchetto ms-office, l’istallazione del software su di una specifica macchina dove risiedevano sia i dati che il programma stesso e il fatto di poter accedere via rete all’applicativo con un maggior livello di fruibilità rispetto al passato. Il server web, il linguaggio di programmazione e il dbms erano di tipo open source e quindi a costo zero. In realtà il fatto che un prodotto sia open source non significa che implicitamente sia a costo zero, ma in questo caso fu ottenuta questa corrispondenza. Durante lo sviluppo in PHP del programma è stato possibile valutare quali strumenti erano necessari per poter soddisfare i punti 4. e 5..

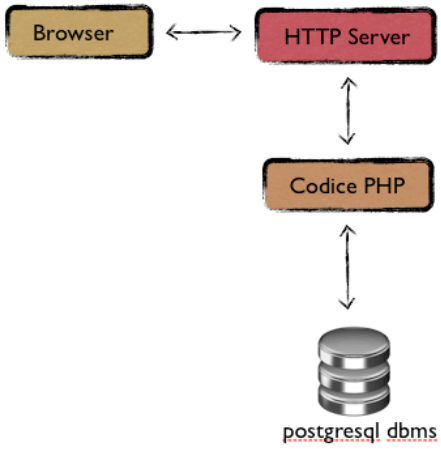
1.3 Organizzazione delle risorse utilizzate per la realizzazione del programma in PHP.

Come mostrato in figura 1.1 sono stati utilizzati il server http apache, il dbms postgresql, il linguaggio di programmazione PHP e la libreria JQuery ed alcuni suoi plugin come treeview e JqGrid. Il codice php è stato strutturato sotto forma di codice in funzione dei dati da gestire e da una libreria di carattere generale nella quale sono state scritte funzioni per:

I. l’autenticazione degli utenti;
II. la profilatura utenti;
III. la definizione parametrizzata di alcune parti degli statement SQL ORDER BY e WHERE;
IV. la gestione dei contenuti della variabile di sessione necessari per sopperire alle note carenze del protocollo http;

V. la validazione dei dati;

Per quanto riguarda la validazione dei dati è stata scelta l’impostazione di effettuare la validazione lato server; in questo modo il processo di convalida dei dati, prima che questi vengano gestiti dalla parte del programma che effettua le modifiche nel database, è svolto e gestito in un solo punto del sistema. Non sono stati quindi implementati due sistemi di validazione: lato client e lato server, che avrebbero comportato due implementazioni distinte. La validazione lato client è non del tutto sicura in quanto l’utente potrebbe eludere i processi di validazione provocando potenziali anomalie sul funzionamento del programma.



	http server	php	dbms	JavaScript (Browser)
Autenticazione		X	X	
Profilatura		X	X	
Logica del Programma		X		
Gestione sessioni	X	X		
Accesso ai dati		X	X	
Validazione		X	X	
Visualizzazione		X		X

Figura 1.1 schema generale del programma.

Sia la validazione dei dati che l’autenticazione degli utenti e la loro profilatura, in questa implementazione, sono legate al dbms utilizzato. Non avendo infatti disponibile un’architettura che permettesse di essere indipendenti dal dbms è stato necessario utilizzare le sue funzionalità con i limiti di avere utilizzato il sistema di accounnting del dbms; per la validazione sono state utilizzate le tabelle di sistema del DBMS che contengono le definizioni dei campi di tutte le tabelle del database. Tutto ciò ha portato a una forte interdipendenza dei vari livelli dell’applicativo,

nel senso che, l’adozione in tempi successivi alla realizzazione del programma, di un dbms differente da quello adottato in origine, avrebbe comportato interventi significativi nel codice a livello di autenticazione, profilatura e validazione.

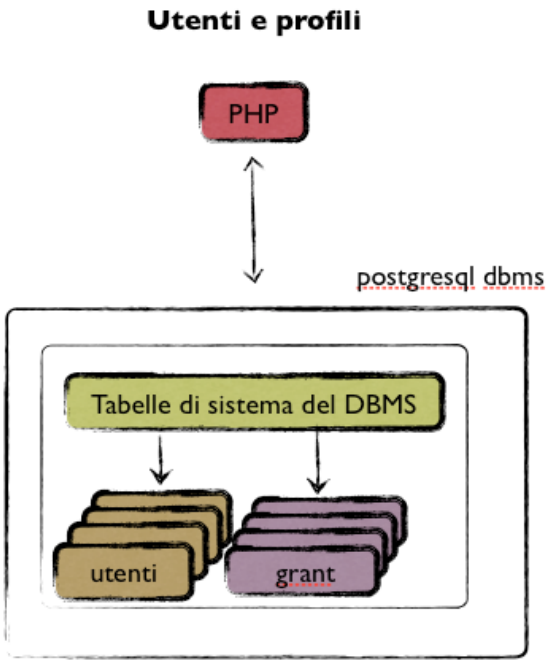


Figura 1.2 autenticazione e autorizzazione gestite tramite dbms

L’utilizzo del programma per più società comporta l’implementazione di un database per ogni società, mentre per le funzionalità di autorizzazione , è necessario l’utilizzo dei GRANT del dbms.

1.4 Javascript

Per poter sviluppare la parte di visualizzazione dati, in modo agevole ed intuitivo, è ormai indispensabile ricorrere a risorse disponibili in javascript che, in alcuni casi, utilizzano AJAX per interloquire con il server. E’ fortemente consigliato utilizzare librerie javascript già disponibili in rete in quanto già utilizzabili e soprattutto compatibili con la maggior parte dei

browser. La libreria utilizzata è JQuery con i plugin treeview e Jqgrid. Questo tipo di soluzione ha portato all’ottimizzazione del traffico di rete fra browser server con la possibilità di utilizzare il programma anche in contesti dove la banda di rete è estremamente limitata.

1.5 Jquery

Jquery è una libreria javascript sviluppata e realizzata da Jhon Resig e pubblicata nella versione 1.0 nel 2005. La struttura delle API, ben documentata, ha permesso una notevole diffusione del suo utilizzo e sviluppo di moltissimi plugin disponibili al sito http://www.jquery.com. Oltre alla sua struttura, progettata con l’idea “Write less, do more” [Res-09] , jquery ha una serie di caratteristiche che hanno facilitato la sua diffusione:

- libreria open source con licenza MIT e GPL;
- estremamente contenuta nelle dimensioni (circa 120 KB);
- normalizza le differenze di compatibilità fra i vari browser;
- strutturata in modo da essere efficiente;
- possibilità di sviluppare in modo agevole plugins;
- progettata in modo da evitare conflitti con altre librerie javascript;
- utilizzata da organizzazioni di livello internazionale (IBM, Microsoft, Dell, CBS, Bank of America, integrata in alcune implementazioni JSF, ecc.);
- possibilità di integrare la realizzazione di plugin con specifiche librerie come ad esempio Jquery User Interface;

1.6 JqGrid

Jqgrid è un plugin disponibile all’URL http://www.trirand.com/blog/, permette la gestione in formato tabellare di informazioni, comprese le funzionalità di inserimento, modifica, cancellazione, ricerca, ordinamento ecc. Per come è stato strutturato, questo plugin, implementa il criterio dell’ottimizzazione del traffico di rete, prevedendo una serie di parametri che possono essere passati al server per ottenere solo i dati da visualizzare, indipendentemente dalla dimensione, in termini di numero di record assoluti della tabella. Il formato dati, supportato per lo scambio di informazioni con il server è XML o JSON, sono implementati una serie di eventi e metodi che permettono di gestire l’attività dell’utente, compresa l’integrazione con Jquery User Interface per la configurazione dell’aspetto grafico dell’oggetto tabella che verrà visualizzata dal Browser.
Le immagini successive mostrano il codice e il relativo aspetto grafico per la gestione anagrafica dei giocatori.

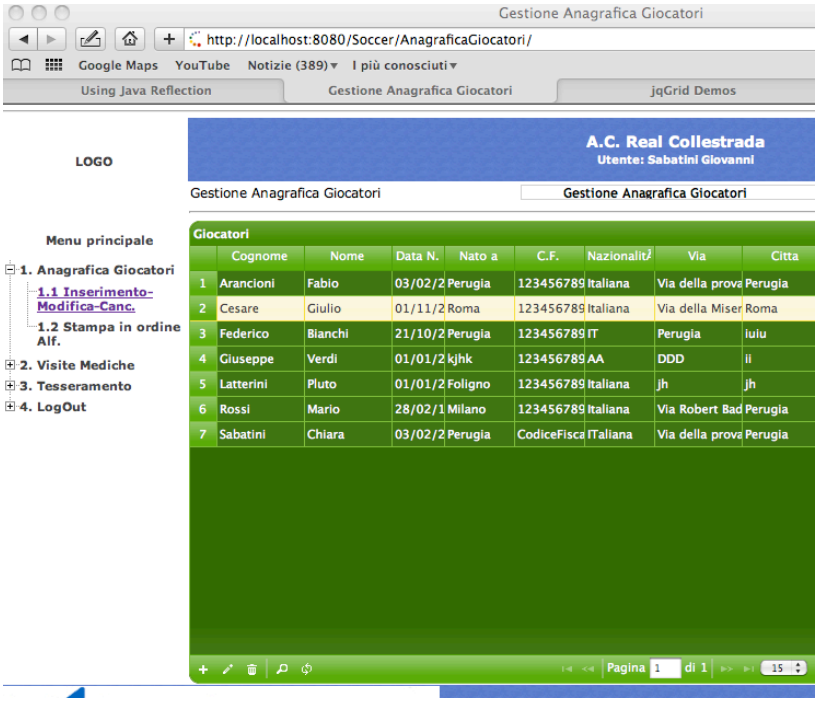


Figura 1.3 esempio interfaccia utente

L’aspetto grafico è relazionato alla configurazione della libreria JQuery UI, mentre si possono facilmente vedere gli oggetti che permettono all’utente di poter interagire con le informazioni in modo efficace da una sola pagina web.

In questo esempio la tabella anagrafica è costituita di soli 7 record, viste però le impostazioni indicate in basso a destra se anche la tabella fosse costituita da un numero elevatissimo di record, verrebbero richiesti al server solo 15 record alla volta con la conseguente ottimizzazione del traffico di rete generato. In figura è riportato un estratto del listato javascript necessario ad implementare quanto visto nell’immagine precedente.

```
.....
                        //alert("Inserimento effettuato con successo");
                        closeAfterAdd : true;
                    }
            jQuery(document).ready(function(){
 jQuery("#list").jqGrid({
   url:'/Tevere/AnagraficaGiocatori/AnagraficaGiocatori.php',
   datatype: 'xml',
   mtype: 'POST',
      colNames:['ID','Cognome', 'Nome','Data N.','Nato a', 'C.F.','Nazionalit?l ','Via','Citta',
'CAP', 'Prov.', 'Tel.', 'Cell.', 'email' ],
   colModel :[
    {name:'id_giocatore', index:'id_giocatore', width:25, hidden: true},
            {name:'cognome', index:'cognome,nome,datadinascita', width:100, align:'left',
editable:true},
                {name:'nome', index:'nome,cognome,datadinascita', width:100, align:'left',
editable:true},
         {name:'datadinascita', index:'datadinascita,cognome,nome', width:60, align:'right',
editable:true},
    {name:'luogodinascita', index:'luogodinascita', width:80, align:'left', editable:true},
    {name:'codicefiscale', index:'codicefiscale', width:80, align:'left', editable:true},
           {name:'nazionalita', index:'nazionalita,cognome,nome,datadinascita', width:80,
align:'left', editable:true},
    {name:'via', index:'via', width:100, align:'left', editable:true},
            {name:'citta', index:'citta,cognome,nome,datadinascita', width:100, align:'left',
editable:true},
    {name:'cap', index:'cap', width:35, align:'left', editable:true},
    {name:'provicia', index:'provincia', width:35, align:'left', editable:true},
    {name:'telefono', index:'telefono', width:100, align:'left', editable:true},
```

```
    {name:'cellulare', index:'cellulare', width:100, align:'left', editable:true},
    {name:'email', index:'email', width:100, align:'left', editable:true, formatter:'email'}
  ],
  pager: '#pager',
  rowNum: 15,
  rowList: [15,30,45],
  sortname: 'cognome',
  //sortorder: 'desc',
  viewrecords: true,
  rownumbers: true,
  overrows: true,
  editurl:"/Tevere/AnagraficaGiocatori/Modifica.php",
  width: 900,
height: 350,
  caption: 'Giocatori'
});
jQuery("#list").jqGrid('navGrid','#pager',{edit:true,add:true,del:true},
      // options
      {
            afterSubmit: function asubmit (response,postdata) {
                  var rc=true;
                  var message=response.responseText;
                  if(message == "OK\n")
                  {
            else
                  {
                        //alert("Errore inserimento dati.");
                        rc=false;
                  }
                  //alert("afterSubmit");
                  return[rc, message, ""];
                }
      },
      {
            afterSubmit: function asubmit (response,postdata) {
                  var rc=true;
                  var message=response.responseText;
                  if(message == "OK\n")
                  {
                        //alert("Inserimento effettuato con successo");
                        closeAfterAdd : true;
.......
<td height="290" valign="top" >
      <table id="list" class="scroll"></table>
      <div id="pager" class="scroll" style="text-align:center;"></div>

</td>

.........
```

Senza andare nel dettaglio, l’impostazione della tabella è piuttosto intuitiva, da notare che si possono personalizzare l’ordine dei campi da visualizzare, i criteri di ordinamento e come interagire con il server per la richiesta e la modifica dei dati. Le operazioni di modifica sono gestite ad eventi con la possibilità di notifica da parte del server di messaggi di errore quali ad esempio il processo di validazione dei dati non andato a buon fine ecc.

1.7 Fpdf

La generazione dinamica di documenti in formato PDF rappresenta la soluzione per la stampa di report relativi ai dati dell’applicazione. La libreria utilizzata per questo servizio è fpdf (http://www.fpdf.org/) con licenza di tipo MIT e con una serie di funzioni che, seppur non estremamente sofisticate, permettono di generare, in maniera semplice ed intuitiva, documenti in formato PDF.

1.8 OpenVpn

La versione php del programma rappresenta il primo approccio all’implementazione del paradigma “Internet” per l’utilizzo dell’applicativo. Oltre alla scontata autenticazione web, implementata tramite semplice interfaccia di login, è stata testata una soluzione migliorativa di accesso al servizio, anche se limitativa all’aumentare del numero di utenze.
Vista la natura dei dati estremamente sensibili, in particolare per il fatto che si tratta di dati di minori, è stato implementato un sistema di accesso ad una rete vpn prima di poter accedere al servizio, con il vantaggio di non esporre il server web direttamente sulla rete ma, al contrario, renderlo pubblico solo per coloro che fanno parte della rete vpn. Anche in questo caso, come per le altre risorse, è stato adottato il criterio di utilizzare software open-source di libero utilizzo e la scelta è ricaduta su OpenVpn.

Si tratta sostanzialmente di una soluzione che non si basa su IPSec, ma al contrario, sfrutta le note librerie OpenSSL, utilizzate anche da parte dei maggiori server http, che tramite network adapter virtuali, disponibili ormai su tutti i S.O. (Es. accesso remoto su windows Xp), permette di implementare a livello applicativo una rete vpn, compresa la disponibilità di client per i più diffusi sistemi operativi che oggi il mercato offre. In questo modo il servizio esposto su Internet è il servizio vpn che può essere acceduto soltanto da quegli utenti che sono in possesso dei certificati rilasciati dal server OpenVpn. Vista l’esperienza estremamente positiva avuta con questo pacchetto ed il lavoro di approfondimento necessario per una comprensione esaustiva del protocollo, viene data una descrizione più dettagliata del prodotto.

Le due soluzioni che costituiscono un riferimento per coloro che si accingono ad implementare un'infrastruttura vpn, sono: openvpn e ipsec che, essendo un un protocollo definito e implementato a partire dagli anni '90 è supportato dalla gran parte dei produttori di apparati di rete e sistemi operativi. IpSec presenta comunque alcune limitazioni, ad esempio complessi meccanismi di autenticazione insiti nel protocollo, necessità di utilizzo di IP statici e difficoltà nell'implementazione di vpn in presenza di rete in NAT.

La soluzione openvpn ha caratteristiche del tutto diverse; infatti, per poter essere implementata non necessità di modifiche ai sistemi operativi in quanto non gira in kernel mode come ipsec, utilizza le interfacce di rete di tipo tunnel presenti ormai su tutti i sistemi e gira in user space. Openvpn non è un protocollo ma un applicativo che accede alle interfacce di rete di tipo tunnel come se fossero dei file e quindi può effettuare operazioni di lettura e scrittura su di esse. I dati vengono cifrati mediante tecniche note e consolidate utilizzate ormai da anni per l'implementazione; ad esempio, di siti web sicuri (SSL), protocollo ssh ecc. Dal punto di vista della sicurezza entrambi i sistemi sono affidabili.

La configurazione di OpenVpn prevede la logica client-server: il client apre la connessione verso il server tramite un tunnel cifrato.
Dalla versione 2.0 di openvpn è possibile avere più di un client contemporaneamente connessi ad un server.
E' possibile utilizzare due tipi di interfacce di tunneling, l'intefaccia tunx che garantisce la connessione punto-punto esattamente come avviene per una connessione PPP e l'interfaccia tapx che garantisce sempre una connessione punto-punto ma lavora in bridged-mode con le altre interfacce di rete in modo da poter rendere disponibili i servizi locali al tunnel.

Openvpn è disponibile al sito http://www.openvpn.org , nelle distribuzioni linux è presente come pacchetto.

1.8.1 OPenVpn generazione delle chiavi

Dalla versione 2.0 di openvpn è possibile utilizzare l'infrastruttura PKI (Public Key infrastructure) che consiste in:

- un certificato conosciuto come chiave pubblica e una chiave privata per ogni client e server;
- un master certification Authority necessario per effettuare la certificazione delle chiavi pubbliche;

Il meccanismo di autenticazione si basa sul principio dell’autenticazione bidirezionale: sia client che server hanno disponibili il certificato master della CA, con il quale sono state firmate le rispettive chiavi pubbliche, e quindi sono in grado di poter verificare l’autenticità delle stesse proposte nella fase di autenticazione. Se la fase di autenticazione termina con successo, il sistema di encryption asincorno viene utilizzato solo per scambiare una chiave di encryption che sarà poi usata sia dal client che dal server per effettuare la codifica dei dati connessione.

Openvpn, nella sua distribuzione, prevede dei tool per generare l'infrastruttura pki in modo da poter implementare la configurazione client-server senza la necessità di richiedere la firma della chiave pubblica ad un certification authority ufficiale. Ecco i passi:

- nella home di openvpn package troviamo la cartella easy-rsa
- creazione del certificato o meglio della certification authority:
  - . ./vars inizializzazione delle variabili di ambiente (vedi script);
  - ./clean-all viene cancellato il contenuto della cartella keys che conterrà certificati e chiavi pubbliche e private e chiavi pubbliche certificate;
  - ./build-ca viene generato il master Certificate Authority (CA) certificate & key, alcune informazioni possono essere preimpostate in vars. Ecco un esempio:

    The final command (build-ca) will build the certificate authority (CA) certificate and key by invoking the interactive openssl command:

    ai:easy-rsa # ./build-ca
  - Generating a 1024 bit RSA private key
- Creazione delle chiavi del server e certificazione della chiave pubblica
  - ./build-key-server <nome server> Come per il precedente certificato, sono richieste alcune informazioni, compresa la richiesta di certificare la chiave pubblica.
- Creazione delle chiavi dei client e certificazione della chiave pubblica (come sopra)
  -

- <nome client 1>
  build-key <nome client 2>
  build-key <... nome client n ...>

- Per openvpn server per linux/unix va creato anche "Diffie Hellman parameters"
  - ./build-dh
  - esempio di output del comando:
    ai:easy-rsa # ./build-dh
  - Generating DH parameters, 1024 bit long safe prime, generator 2
  - This is going to take a long time

La configurazione sia lato client che server necessita di un file di testo dove sono specificate le opzioni di connessione. Tutte le implementazioni disponibili per S.O. come Windows, OSX e linux adottano, per i file di configurazione, lo stesso formato.

1.9 Funzionalità del programma

L’analisi del programma è stata effettuata in collaborazione con il Dott. Piero degli Esposti, Direttore Tecnico della società calcio Tevere, con esperienza ventennale nel settore giovanile, con il quale sono state fissate le funzionalità principali, ma come vedremo, si potrebbero aggiungere altre funzioni, vista la caratterizzazione di tipo Enterprise che la procedura ha subito .

Si elencano le funzionalità del programma:

I. gestione anagrafica giocatori;
II. gestione anagrafica della società;
III. gestione visite mediche;

IV. gestione tesseramento;
V. gestione formazioni;
VI. gestione calendario;
VII. gestione magazzino.

In questo lavoro di tesi non viene analizzato il codice scritto per implementare le funzionalità sopra elencate, ma gli aspetti fondamentali per applicazioni, che si basano su protocollo http e che sono fruibili dalla rete Internet.

Capitolo 2 - J2EE principi base

2.1 Introduzione

La nascita del protocollo http (Hypertext Transfer Protocol) fu sviluppato per consentire l'accesso ad informazioni di tipo statico, si trattava quindi di un protocollo molto semplice e di tipo stateless.

A questo stato iniziale venne, in breve tempo, aggiunta la possibilità di gestire e consultare contenuti dinamici che hanno portato allo sviluppo di siti web di notevole complessità con l'implementazione di servizi di tipo commerciale. Le varie tecnologie che via via si sono evolute, per consentire lo sviluppo di applicazioni fruibili tramite protocollo http, sono molteplici, di varia tipologia e hanno determinato un proliferare di sistemi e soluzioni. Fra queste troviamo la possibilità di poter sviluppare applicativi per la gestione di contenuti dinamici di pagine web utilizzando codice, scritto in linguaggio Java.

In questo capitolo verranno illustrate le caratteristiche dello standard J2EE adottato per la migrazione della procedura, oggetto di questo lavoro di tesi.

2.1 Servlet

Le servlet sono porzioni di logica scritte in Java, che, se invocate, generano una risposta o effettuano delle azioni [Chop-Li-Gen]. Rispetto al vecchio modello dei programmi CGI, questa soluzione risolve i seguenti problemi:

1. a differenza del vecchio schema, dove il sistema operativo aveva l'onere di istanziare e terminare i processi associati alle richieste di esecuzione di codice per la generazione di contenuti dinamici, con

conseguente overhead e impatto negativo sulle prestazioni, questa soluzione prevede l'utilizzo di una sola istanza JVM;

2. ad una richiesta di esecuzione di una servlet, la JVM effettua il load delle classi necessarie alla sua esecuzione. Nel caso in cui servlet utilizzino classi comuni queste saranno caricate in memoria dalla JVM una sola volta;

3. nelle specifiche di definizione delle servlet viene colmata la lacuna del protocollo HTTP relativa alla gestione degli stati;

4. le servlet implementano un'interfaccia standard che definisce il ciclo di vita e un insieme di metodi che possono essere invocati per il suo utilizzo;

5. le servlet non hanno un metodo main(), ma sono eseguite sotto il controllo di un'altra applicazione java denominata servlet container. Quando un server web identifica una richiesta di esecuzione di una servlet, il server web non esegue direttamente la servlet, ma indirizza la richiesta di esecuzione al servlet container dove la servlet è stata pubblicata. E' quindi il servlet container che prende in gestione la richiesta, esegue il codice e restituisce il risultato al server web.

I servlet container possono essere utilizzati secondo tre principali configurazioni come riportato in figura 2.1. IL web server può essere implementato nella JVM che gestisce il servlet container, vi sono soluzioni dove il server web è gestito come processo direttamente dal sistema operativo e soluzioni dove il server web ed il servlet contanier si trovano su sistemi distinti connessi in rete.

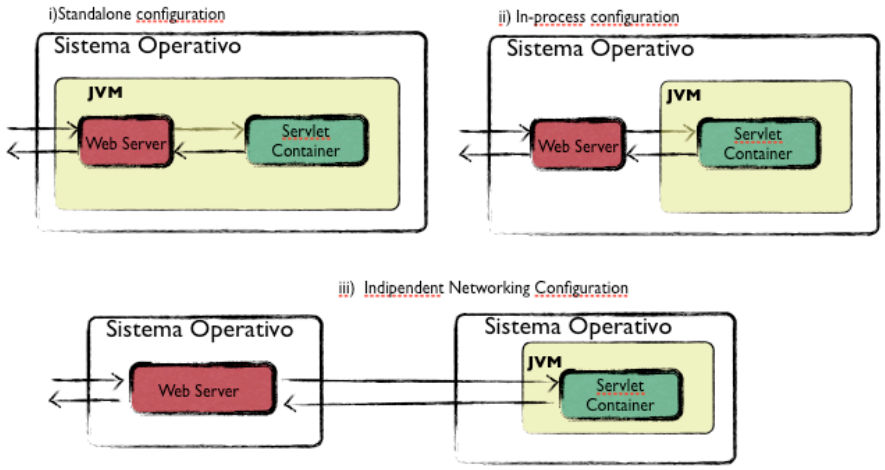


Fig. 2.1 Schema configurazioni principali server web - servlet container

Oltre a quanto riportato in figura 2.1, il server container svolge anche altre funzioni che permettono di semplificare l'implementazione del codice della servlet, ottimizzare l'uso delle risorse, ridurre i rischi di bug nel codice, gestire il ciclo di vita della servlet comprese le funzionalità di autenticazione ed autorizzazione, supporto per il multithreading, supporto per le pagine JSP e per la gestione delle sessioni. In realtà quando si realizza una pagina jsp e quindi si inserisce codice java in una pagina html, il servlet container si prende carico di convertire la pagina in una servlet che poi sarà eseguita dal container stesso. Una pagina JSP è una servlet scritta in una modalità di più alto livello rispetto ad una servlet vera e propria. Questo principio viene applicato anche in altri contesti, questo perchè lo sviluppo degli standard JSP, SERVLET, J2EE hanno portato alla progettazione di specifiche per collocare le diverse tipologie di attività in strati applicativi ben definiti (server web, servlet container, servlet, ecc.) con il risultato di poter eseguire la propria applicazione (servlet) su servlet container diversi. Se poi si considera che i servlet container sono eseguiti a loro volta in una JVM, implicitamente viene a determinarsi l'indipendenza fra sistema operativo, servlet container e servlet.

2.2 Modello MVC

Limitarsi a sviluppare contenuti dinamici di un'applicazione web tramite l'utilizzo di pagine JSP comporterebbe una serie di problematiche dovute alla contemporanea presenza nel codice di due tipologie di informazioni completamente diverse e cioè costituite dalla componente HTML e da istruzioni Java con notevoli difficoltà per la gestione dello sviluppo, del debugging e delle competenze, che in questo caso, dovrebbero essere di tipo trasversale per coloro che sviluppano l'applicativo. Il modello che tende a risolvere questo problema prende il nome di **M**odel-**V**iew-**C**ontroller che prevede la separazione dell'applicazione in tre parti fondamentali:

1. la logica del programma (Controller);
2. la gestione della persistenza dei dati (Model);
3. la visualizzazione delle informazioni(View).

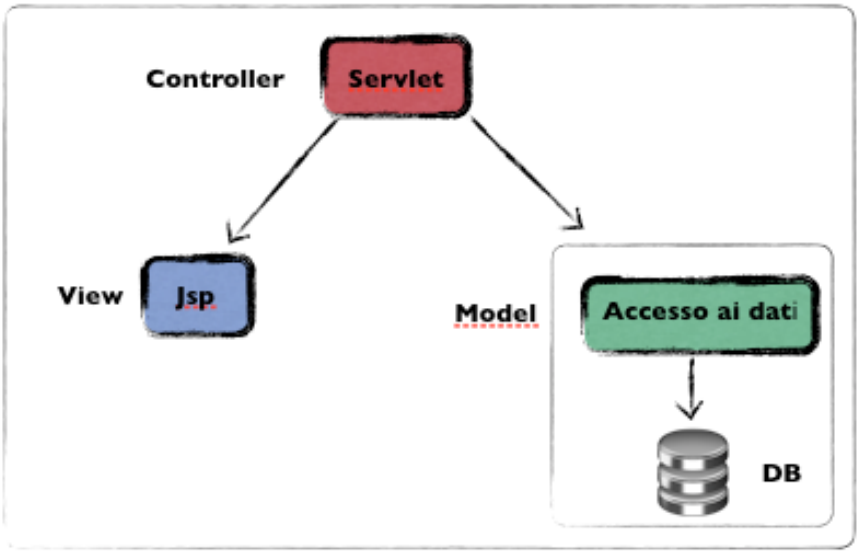


Fig. 2.2 Schema MVC

2.3 Esempio MVC

Nell'esempio riportato in figura 2.3 e 2.4 viene illustrata la struttura di un'applicazione che accetta l'inserimento dei dati (nome, cognome e età) tramite un form ed effettua il salvataggio delle informazioni su file di testo. Lo strumento di sviluppo utilizzato, sia per la realizzazione di questo esempio che per la migrazione dell'applicativo è l'IDE eclipse versione "Helios" JavaEE con plugin per Jboss 5.0 e Ldap. Il progetto FirstJSP è strutturato in modo che la pagina html, per l'inserimento dei dati tramite il form, invoca la servlet denominata simpleController la quale gestisce i dati passati come argomento, istanzia gli oggetti necessari per la memorizzazione dei dati su un file di testo e invoca la pagina JSP utilizzata per la visualizzazione e l'esito delle operazioni svolte all'utente finale. La struttura MVC presentata in questo esempio è inserita in un contesto secondo uno schema standard che l'IDE genera in funzione del tipo di progetto che si va a realizzare.

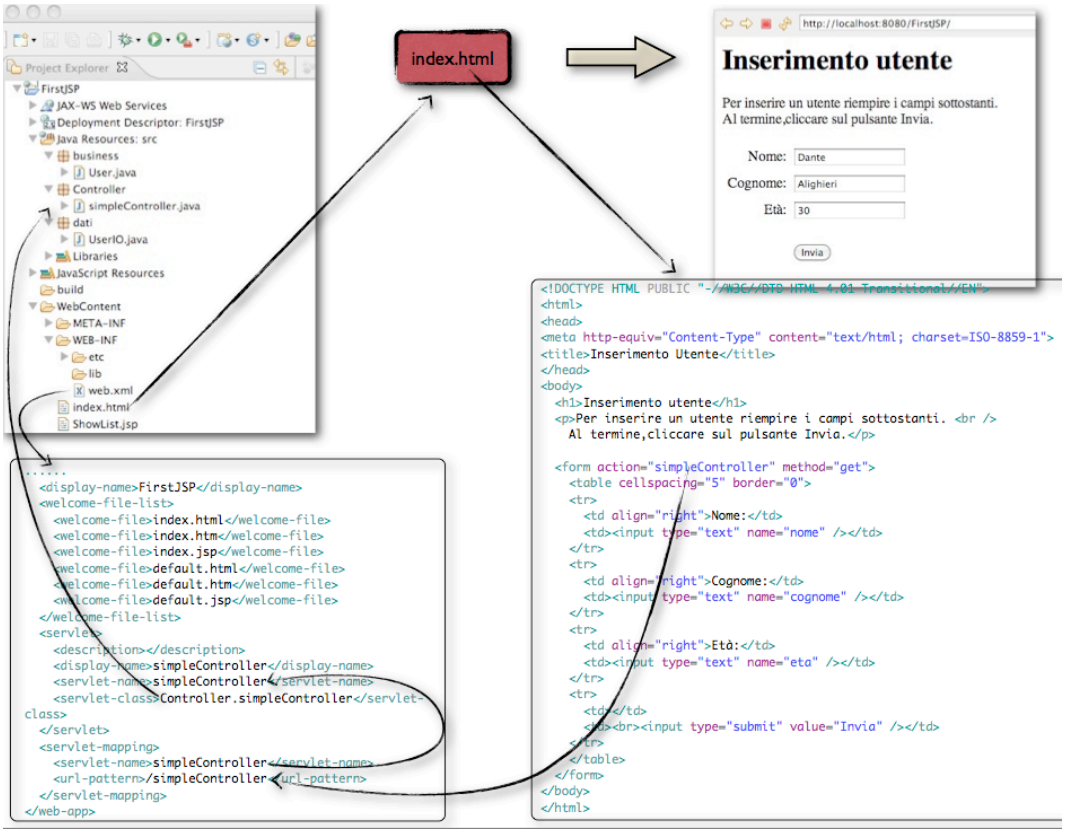


Fig. 2.3 Rappresentazione grafica di un esempio che implementa il modello MVC.

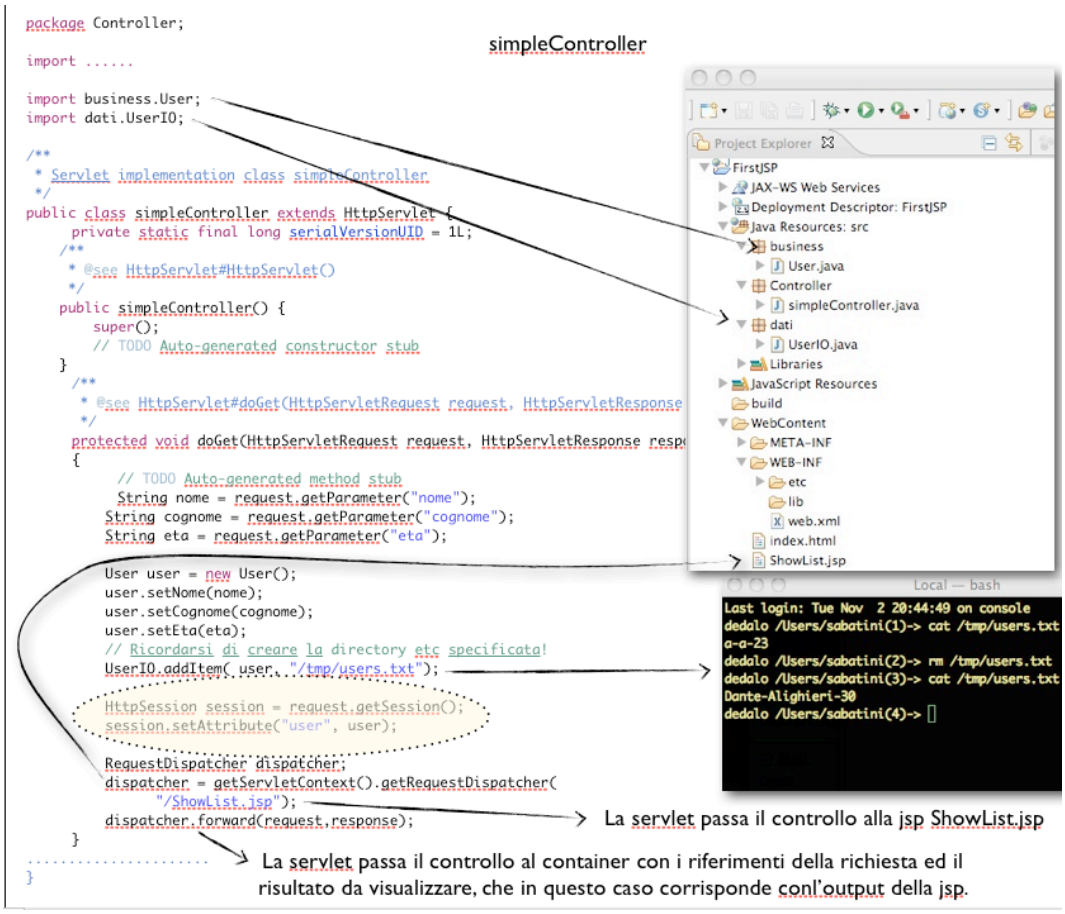


Fig. 2.4 Rappresentazione del controller che implementa la logica dell'esempio.

Il primo impatto che si ha osservando la struttura in figura 2.3 e 2.4 è quello di un'apparente complicazione dell'architettura rispetto, ad esempio, alla semplicità strutturale di applicativi web scritti con altre tecnologie, php, perl, ecc.

2.4 Java 2 Enterprise Edition

La tecnologia J2EE fu introdotta da Sun Microsystem nel 1999. Dalla sua prima uscita si è notevolmente evoluta e consolidata diventando di fatto uno degli standard utilizzati per lo sviluppo di applicazioni di tipo

"Enterprise". E' importante ricordare il contributo che Sun Microsystem ha dato in modo significativo allo sviluppo nelle tecnologie informatiche, basta ricordare l'implementazioni di NFS, NIS, Java, Sun Directory Server, ecc..

J2EE implementa il modello MVC organizzato secondo quattro livelli principali:

1. livello per la visualizzazione e gestione degli eventi che possono essere attivati dall'utente finale;
2. il livello per la gestione delle richieste dei client http verso i componenti servlet e jsp;
3. il livello che implementa la logica applicativa sviluppata per componenti (**E**nterprise **J**ava **B**ean);
4. livello che gestisce la persistenza dei dati verso i dbms.

La scelta di suddividere l'infrastruttura in livelli logici distinti, raggruppati per tipologia omogenee di funzionalità conferisce al modello la caratterizzazione di tipo enterprise. La possibilità cioè di poter implementare processi per la gestione aziendale di notevole complessità e di diversa tipologia: dalle aziende produttive, alle istituzioni scolastiche a quelle governative in genere ecc.

L'implementazione dei quattro livelli sopra citati sfrutta una delle peculiarità della programmazione ad oggetti, cioè il riutilizzo del codice. In J2EE sono presenti tutta una serie di tecnologie già sviluppate in passato per altri scopi e riutilizzate in questo ambito. Tutto ciò ha prodotto implementazioni di notevole complessità dove sono integrate fra loro componenti molto complesse che costituiscono le parti fondamentali dello standard J2EE.

Chiunque si accinge ad effettuare un'indagine conoscitiva di questa tecnologia avrà un impatto abbastanza forte, vista la complessità dei suoi componenti. Il quesito quindi che si pone è l'opportunità di sviluppare applicativi secondo questo standard. Pensare di realizzare un

programma che al suo interno abbia tutte le componenti necessarie per poter essere considerato di tipo enterprise, comporterebbe uno sviluppo estremamente complesso che andrebbe oltre la logica del programma necessaria a implementare quanto richiesto per la gestione dei processi aziendali. Si deve quindi considerare che la realizzazione di applicativi in J2EE presuppone una netta scissione fra l'implementazione dei processi di gestione della logica del programma (business logic) dalle funzionalità che sono necessarie per garantire scalabilità e robustezza dell'applicativo. Il risultato finale è quello di utilizzare un'infrastruttura tecnologica che implementi quelle funzionalità non specifiche dell'applicativo che prende il nome di application server. Ecco quindi che rispetto a quanto descritto nel capitolo uno, dove la versione in php del programma conteneva al suo interno tutte le funzionalità necessarie, vista l'assenza di una infrastruttura capace di assolvere specifiche funzioni e quindi semplificare lo sviluppo e la complessità del codice. Dal punto di vista figurativo, un application server si potrebbe immaginare come un enorme macchinario con delle parti mancanti, delle quali però si hanno le informazioni per poterle costruire (il proprio applicativo) ed incastrarle nel marchingegno. Una volta completato il quadro, il macchinario inizia a funzionare, senza però averlo costruito per intero.

2.5 Enterprise Java Bean

Una delle parti fondamentali di J2EE è quella costituita dagli Enterprise Java Bean che permettono di poter realizzare componenti distribuiti per applicazioni scalabili e sicure, utilizzando infrastrutture già esistenti e consolidate nel tempo. In figura 2.5 è illustrato lo schema di riferimento dello sviluppo a più livelli di un applicativo (*multi-tier*).

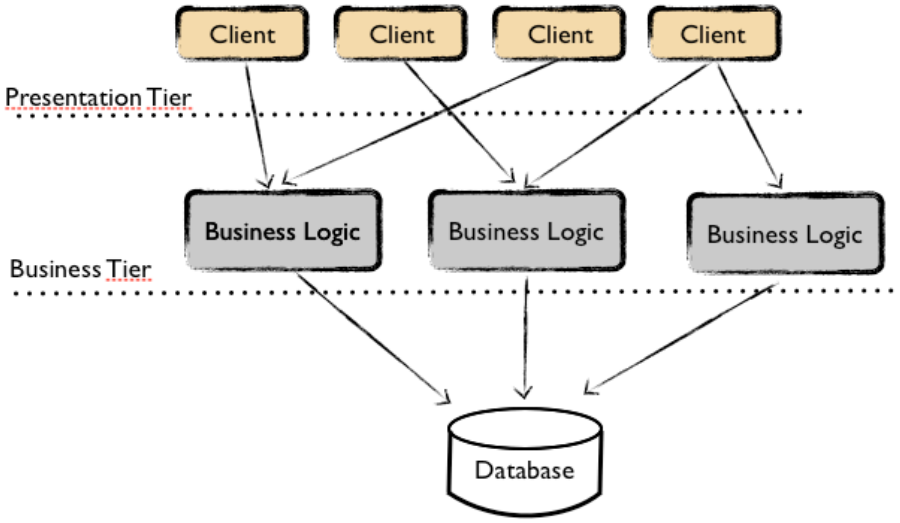


Fig. 2.5 struttura multi livello di un'applicazione.

Il modello rappresentato in figura è piuttosto generico, è quindi necessario approfondire alcuni aspetti in modo da definire delle caratteristiche più precise rispetto a quanto illustrato:

1. **accesso via rete.** I livelli sono collegati fra loro via rete con l'implementazione di meccanismi per lo scambio di informazioni fra i tre livelli;
2. **load balancing**. Il bilanciamento del carico va gestito in modo da ottimizzare le prestazioni;
3. **trasparent failover**. A fronte di blocchi o crash improvvisi l'erogazione del ser izio deve sempre essere garantita;

4.transactions. Il sistema prevede meccanismi di gestione di persistenza dei dati che garantiscono, anche a fronte di crash o errori inaspettati, l'integrità degli stessi;

5. componet life cycle. La gestione del tempo di vita dei componenti del sistema deve essere gestita in modo completo ed efficiente, in modo da garantire il corretto funzionamento dell'applicativo, qualunque sia il suo livello di complessità;

6. security. Va garantito il corretto funzionamento dei livelli di sicurezza adottati, in modo da evitare accessi e/o modifiche non autorizzati ai dati e ai componenti dell'applicativo;

7. logging e auditing. La gestione dei sistemi di log permette di gestire le informazioni necessarie alla corretta individuazione della esecuzione dei moduli dell'applicativo. Considerato il punto 6. e viste le normative sulla gestione dei log che, in molti casi, in funzione del tipo di informazioni che vengono gestite (L. 196/2003 e successive), diventa di fondamentale importanza l'affidabilità del sistema di logging ed auditing;

8. clustering. Il transparent failover viene garantito con configurazioni in cluster in modo che, a fronte di crash o malfunzionamenti in genere inaspettati, il servizio continua ad essere comunque fruibile all'utenza. Il realtà il clusering può assolvere anche alle funzioni di load balancing già elencate al punto 2.;

9. shutdown e aggiornamento del sistema. Sia le operazioni di manutenzione e/o aggiornamento dei sistemi e procedure deve avvenire senza compromettere il funzionamento dei servizi.

Il framework J2EE, a differenza di altre soluzioni disponibili sul mercato supporta solo il linguaggio Java e quindi, in prima battuta, potrebbe sembrare più restrittivo. In realtà il linguaggio Java è uno dei migliori strumenti per scrivere codice strutturato per componenti con il supporto nativo alla gestione delle interfacce e delle rispettive implementazioni. L'esecuzione del codice nella Java Virtual Machine garantisce sia un elevato livello di sicurezza che di portabilità su molte piattaforme.

La figura 2.6 illustra la struttura d'insieme fra le varie componenti dalla quale è possibile cogliere la suddivisione in strati delle risorse utilizzate per lo sviluppo delle applicazioni.

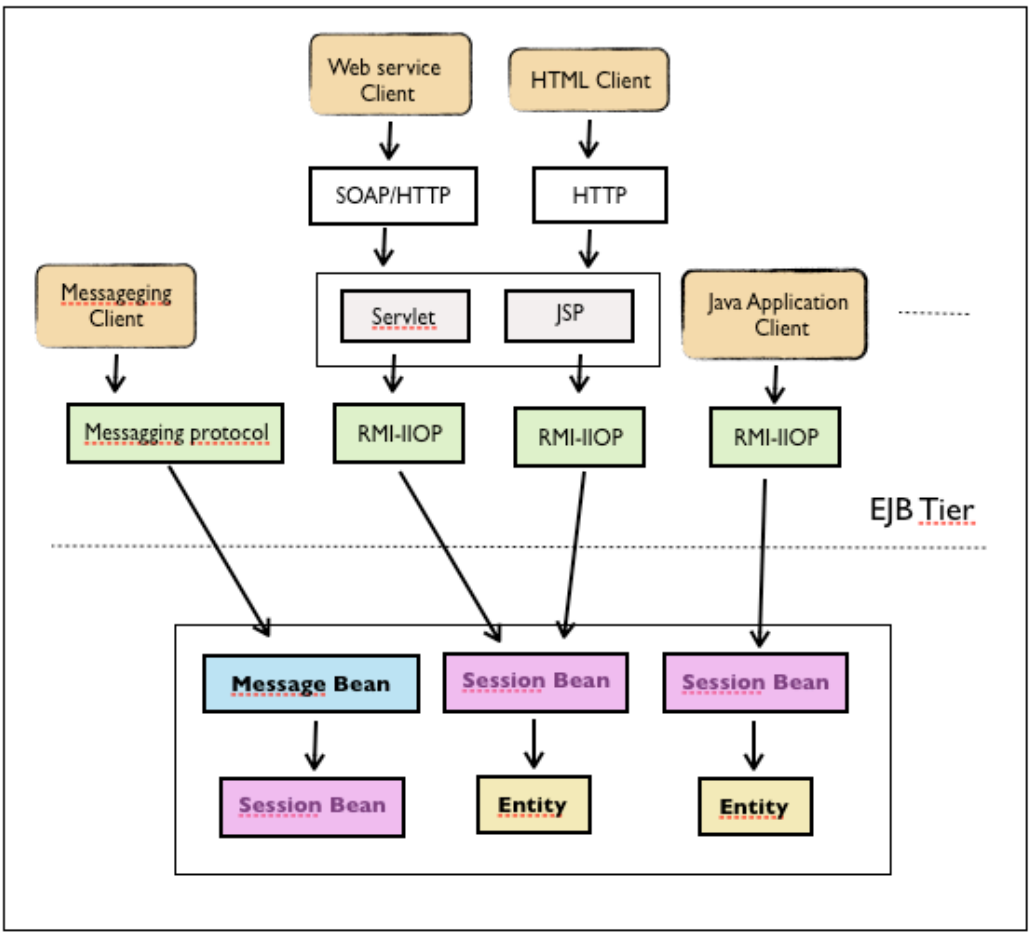


Fig. 2.5 EJB-tier in J2EE.

2.5.1 EJB Container

Gli EJB container sono quelle parti di codice che implementano le specifiche EJB. Sono disponibili sul mercato diversi pacchetti software certificati da SunMicroSystem, fra i principali application server troviamo:

1. IBM WebSphere;
2. Glassfish di SunMicrosystem;
3. Jboss di RedHat;
4. Geronimo di Apache Fondation;
5. Bea Weblogic.

2.6 Session Bean, classificazione

I Session Bean sono dedicati alla logica di business, gestiscono ed implementano la logica dell'applicativo, non gestiscono la persistenza dei dati, non possono essere utilizzati da più client contemporaneamente ed hanno un tempo di vita relativamente breve, in genere per la durata di una sessione.

Quando ad esempio un client richiede determinate operazioni, tramite l'invocazione di un session Bean, a sessione terminata, l'application server procederà alla sua distruzione. I session bean sono strutturati secondo la logica della distribuzione degli oggetti, una delle caratteristiche fondamentali delle specifiche J2EE, tramite l'utilizzo del protocollo Java Name Directory Interface. Grazie a questa tecnologia, è possibile poter istanziare e quindi eseguire codice su una server remoto, come se fosse locale. JNDI implementa un livello di astrazione superiore a RMI (Remote Management Interface) che, tramite l'implementazione dei moduli skeleton e stub permette la distribuzione ed esecuzione degli oggetti fra server collegati in rete. I session bean si suddividono in due categorie: stateless e staefull. I primi non conservano lo stato dei dati ed ad ogni metodo invocato dal client, l'EJB container distrugge l'EJB istanziato e ne crea uno nuovo, senza conservare lo stato dei dati dell'istanza precedente. Questo modello viene implementato dall'application server tramite la gestione di un pool di EJB, che sono utilizzati per erogare la business logic ai client.

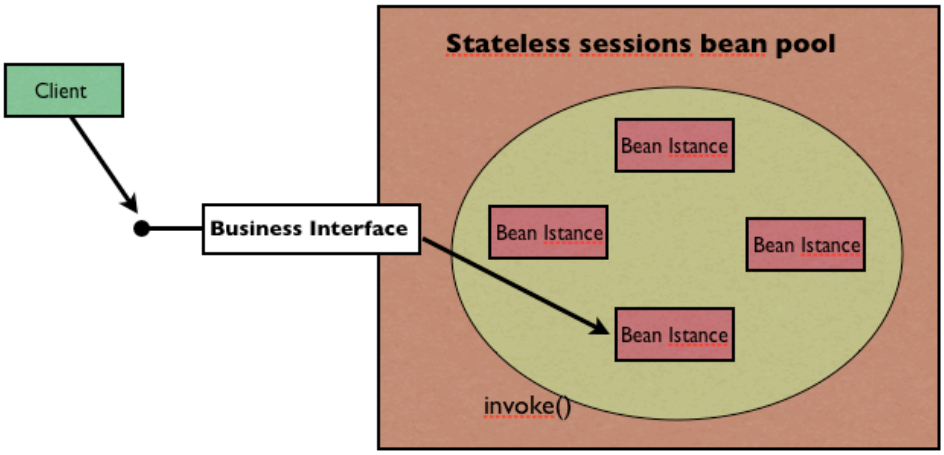


Fig. 2.7 Schema EJB di tipo stateless [Srig-Bro-Silv-2006].

Gli EJB sono strutturati secondo la definizione di un'interfaccia e la loro conseguente implementazione; possono essere di tipo local, se eseguiti in locale, o remote se istanziati ed eseguiti in remoto. Gli EJB di tipo stateless sono particolarmente efficienti, richiedono poche risorse e sono relativamente semplici da implementare.

I session bean di tipo statefull conservano lo stato dei dati per tutta la durata della sessione; un classico esempio potrebbe essere quello dell'implementazione del carrello della spesa, dove, per tutta la sessione è necessario conservare lo stato degli acquisti. Questo tipo di session bean, rispetto ai staless, è più complesso da gestire e richiede maggiori risorse computazionali che, in alcune circostanze, potrebbero portare seri problemi di performance. Al fine di ottimizzare l'uso delle risorse in funzione di istanze EJB session bean stateless, le specifiche J2EE prevedono l'implementazione di procedure di passivation e activation dei session bean. Quando cioè le risorse sono prossime al loro esaurimento, l'application server effettua lo swap su disco delle istanze definite in memoria secondo l'algoritmo Least Recently Used (LRU). Dalla versione

3.0 di EJB è possibile utilizzare la tecnica delle annotazioni che prevede specifici tag per controllare le fasi passivaction ed activation in modo da rilasciare le risorse non serializzabili come ad esempio le connessioni di rete aperte (sockets), file aperti e connessioni verso i database.
Nella firuga 2.8 è rappresentato lo schema generale relativo alla fase di passivation di uno staefull Session Bean.

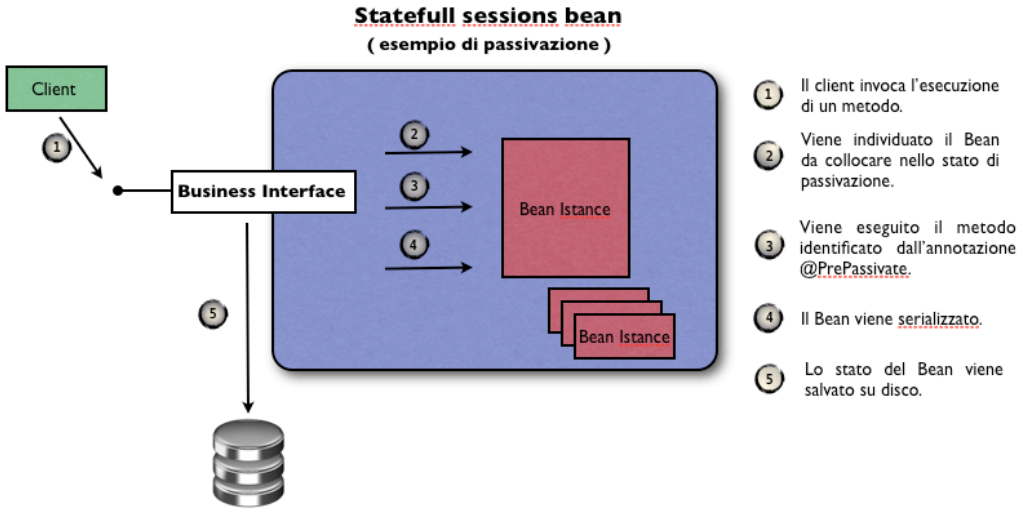


Fig. 2.8 Passivation di un staefull session bean.

La fase di attivazione avviene con logica inversa rispetto a quanto illustrato in figura.
Le specifiche relative agli oggetti EJB permettono di sfruttare molte delle potenzialità del linguaggio Java compreso l'uso delle annotazioni disponibili dalla versione Java 5 in poi. In questo lavoro di tesi non sono state approfondite le caratteristiche degli EJB di tipo Message-Driven ,perchè utilizzabili per lo sviluppo di applicazioni asincrone che non necessitano di una risposta immediata, in quanto non necessari per la migrazione dell'applicativo.

2.7 Entity Bean

Vista la loro importanza, gli entity bean meritano un paragrafo a parte anche perchè non sono degli EJB, non sono compatibili allo standard J2SE e permettono la gestione degli oggetti per la persistenza dei dati. In Java la persistenza dei dati è ottenuta tramite la serializzazione degli oggetti [Bru-06]; l'oggetto viene cioè scomposto in una serie di bit registrati su disco e può essere recuperato effettuando l'operazione inversa alla serializzazione. L'oggetto in memoria può essere salvato come una sequenza di bit. In un contesto enterprise, questa tecnica si rileva insufficiente e inadeguata, basta ricordare le problematiche relative alla ricerca, alla persistenza dei dati ecc.
L'implementazione delle funzionalità necessarie alla gestione del mapping fra oggetti e database può essere realizzata in due modalità:

1. tramite la scrittura di codice che gestisce questo tipo di funzioni. Il programmatore ha quindi l'onere di sviluppare metodi che, tramite API JDBC, effettuano le operazioni necessarie per implementare la sincronizzazione fra oggetti e database. Questo tipo di soluzione implica una complessa gestione di queste funzionalità, dalla realizzazione al loro mantenimento;
2. tramite l'utilizzo di oggetti che rispettano le specifiche JSR 12 (www.jcp.org) che definiscono delle API standard per la gestione del livello di persistenza dei dati e l'astrazione dal tipo di dbms utilizzato.

Questo mapping fra oggetti e database prende il nome di **Object Relation Mapping (ORM)**. I componenti che implementano la persistenza dei dati, ad esempio la persistenza dei dati di un conto bancario, sono denominati **persistent data object**: gli oggetti persistenti sono chiamati **entità**.

Le entità sono strutturate secondo il paradigma POJO e contengono le variabili per la gestione dei campi di una tabella di un database e i metodi per poter effettuare le operazioni di get e set secondo le specifiche dei Java Bean [Bru-05]. Un oggetto POJO viene dichiarato di tipo entità tramite l'utilizzo delle annotazioni.
Le principali differenze fra entità e session bean sono:

1. le entità sono accessibili tramite la propria chiave primaria;
2. hanno uno stato di persistenza;
3. non sono accessibili da remoto;
4. il loro tempo di vita può essere completamente diverso da quello dell'applicazione.

I pacchetti più diffusi che implementano il paradigma ORM sono Hybernate e Toplink. Nel caso dell'Application Server Jboss l'ORM di default è hybernate.

2.8 Entity Manager

Una delle novità più importanti introdotte dalla versione 3.0 degli entity bean è l'introduzione di un'interfaccia per la gestione della persistenza delle entità che permette di gestire tre tipologie di operazioni principali:

1. la gestione del ciclo di vita di un'entità;
2. le operazioni di sincronizzazione fra entità e database;
3. il lookup delle entità e l'esecuzione delle query.

L'Entity Manager API permette di gestire il contesto di persistenza in due modi:

1. **transaction-scoped**. Java EE container usa questo metodo nel caso in cui la business logic è implementata con session bean di tipo

stateless. Il contesto di persistenza termina in concomitanza con la fine della transazione;

2. **extended-persistence**. In questo caso l'entità rimane gestita fino a quando lo statefull session bean non viene cancellato.

Il ciclo di vita di un'entità, che gestisce il contesto di persistenza e la gestione della sincronizzazione con il database, è suddiviso in quattro fasi:

1. **new**. L'entità è stata creata ma non è ancora stata associata a nessun stato di persistenza;
2. **managed**. L'entità è posta nello stato di persistenza tramite la chiamata al metodo persist(). E' garantita la persistenza anche se la sincronizzazione con il database potrebbe essere effettuata in tempi successivi;
3. **detached.** L'entità è persistente ma potrebbe non essere più associata ad un contesto di persistenza;
4. **removed**. L'entità è associata con un contesto di persistenza ed è stata schedulata per essere rimossa dal database.

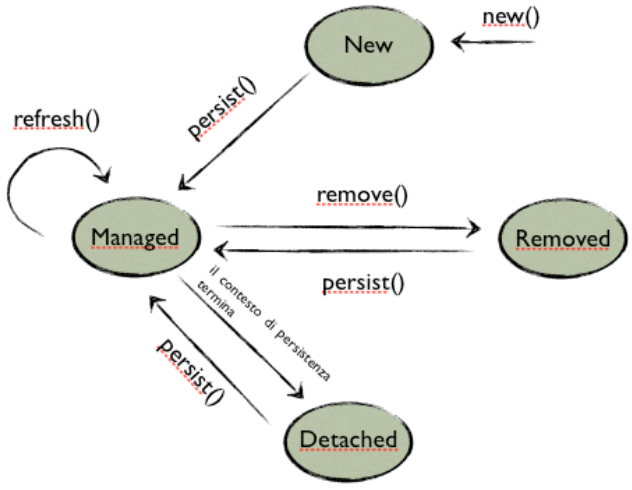


Fig. 2.9 ciclo di vita di un'entità.

In alcuni casi può essere necessario forzare l'aggiornamento del contesto di persistenza in modo esplicito o forzare la sincronizzazione fra entità e database; i metodi setFlushMode(), flush() e refresh() assolvono a queste funzioni.

2.8.1 accesso concorrente e lock

L'accesso concorrente dei dati viene garantito dall'isolamento delle transazioni. Le specifiche *Java persistence* adottano gli algoritmi optimistic locking o esplicite operazioni di READ e WRITE. Nel primo caso le informazioni sono sempre accessibili con il presupposto che non vi siano modifiche concorrenti dello stesso dato. Al momento del *commit* della transazione viene verificato che non vi siano altre transazioni in corso per lo stesso dato; in caso di conflitto, sarà effettuato il *roll-back* della transazione. Nel secondo caso si possono esplicitamente abilitare le operazioni di accesso ai dati sia in lettura che in scrittura.

Come noto, lo standard J2EE è molto complesso. Le caratteristiche delle risorse fin qui descritte sono una piccola parte, ma permettono di poter individuare un quadro d'insieme sufficiente per implementare una semplice applicazione. La figura 2.10 descrive lo schema degli oggetti e delle loro relazioni in un contesto applicativo.

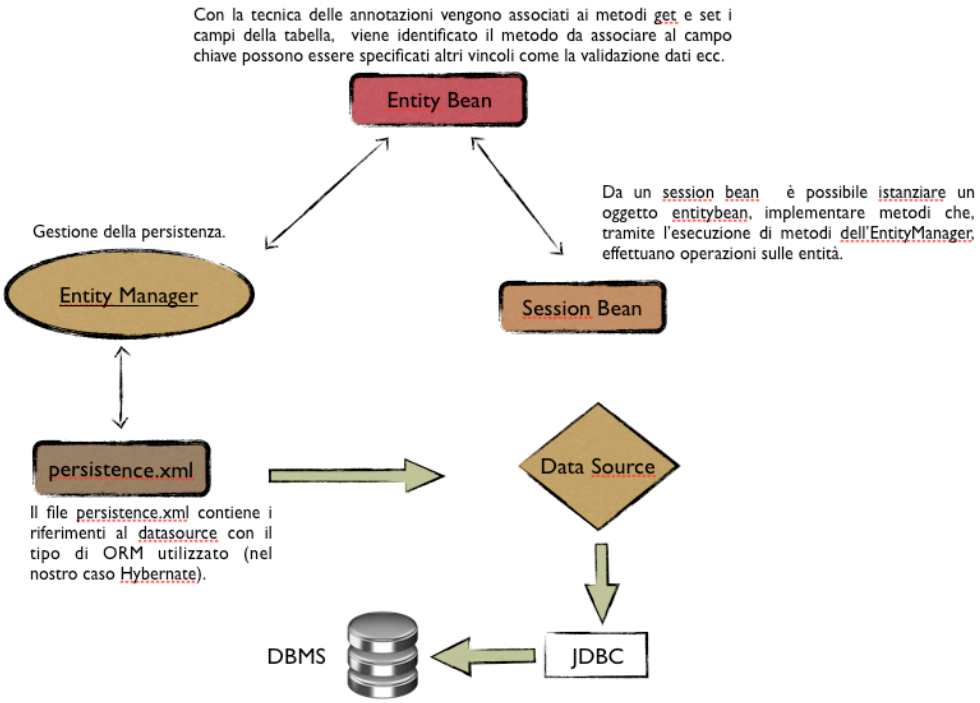


Fig. 2.10 schema generale

L'entity bean è il java bean nel quale vanno definiti i metodi get e set per la gestione dei dati dell'oggetto che, tramite l'utilizzo delle annotazioni, sarà dichiarato di tipo entity. Ogni metodo sarà associato ad uno specifico campo della tabella, si potranno utilizzare le annotazioni per la validazione dei dati e per la dichiarazione del campo chiave della tabella.

L' entity manager in base alle definizioni delle entità, del tipo di ORM adottato e del datasource (la connessione verso il database), garantisce la persistenza dei dati. I metodi dell'entity manager possono essere invocati da un session bean.

Questo è il quadro di riferimento generale utilizzato per la migrazione dell'applicativo oggetto di questo lavoro di tesi.

2.9 Sicurezza

La realizzazione di applicazioni di carattere enterprise che gestiscono complessi meccanismi di elaborazione, necessitano di strumenti per la gestione dell'autenticazione e autorizzazione all'accesso delle risorse. Le problematiche relative alla sicurezza sono complesse da implementare, particolarmente critiche e costituiscono un potenzialie pericolo per accessi non autorizzati ai dati e alle procedure, denominate violazioni del sistema. La valutazione dei rischi di sicurezza è di fondamentale importanza e si basa sul concetto generale di analisi del rischio che esprime il prodotto fra la probabilità che una violazione si verifichi moltiplicato per il danno prodotto. Eliminare le vulnerabilità del sistema è impossibile, si devono quindi utilizzare tutti quei controlli che permettono di ridurre i rischi ad un valore ritenuto accettabile.

In generale, un sistema di protezione è costituito da più componenti ognuno dei quali ha funzioni specifiche. Nel caso in cui un modulo dell'applicativo effettui operazioni di elaborazione e accesso alle informazioni, è necessario che una serie di controlli riducano il rischio di operazioni non autorizzate ad un livello ritenuto sicuro. Nel caso di applicazioni sviluppate con il modello MVC i sistemi di controllo sono espressi dalle seguenti funzioni:

1. **autenticazione**. Verificare l'autenticità dell'utente, l'entità autenticata prende il nome di *principal* [ ];
2. **autorizzazione**. Controllare se l'utente autenticato accede alle risorse secondo i privilegi concessi dal sistema;
3. **protezione dell'integrità dei dati**. Evitare le alterazioni delle informazioni ad esempio quando un utente effettua un'operazione di modifica di un dato, tramite una richiesta pervenuta da una rete non sicura ;

4. **protezione dei dati**. Controllare che modalità di accesso alle informazioni e/o ai moduli del programma siano effettivamente rispondenti alle politiche di sicurezza previste.

Nelle applicazioni web l'autenticazione è supportata da tre meccanismi []:

1. **http basic and digest authentication**. Uno user ID e password, codificati in base64, sono trasmessi dal client al server tramite uno specifico campo del protocollo http. La codifica base64 è estremamente debole e quindi, per aumentare il livello di sicurezza, deve essere combinata con l'implementazione di connessioni sicure SSL (HTTPS);
2. **form based authentication**. Questo metodo si basa sulla spedizione al server delle proprie credenziali tramite la compilazione di un form. In questo caso l'adozione di connessioni SSL diventa indispensabile;

3. **https client authentication**. L'autenticazione si basa sulla verifica dei certificati client server rilasciati da una Certification Authority. Verificata l'integrità dei certificati viene avviata la connessione al servizio richiesto.

Ad autenticazione avvenuta, l'autorizzazione all'accesso alle risorse può essere implementato in due modalità:

1. **declarative Security**. L'autorizzazione all'accesso delle risorse è gestita dal servlet container;
2. **programmatic security**. Nella servlet sono implementati i meccanismi di controllo e verifica delle autorizzazioni.

2.9.1 EJB

Le due fasi che devono essere superate prima che un metodo di un EJB possa essere eseguito sono l'autenticazione e l'autorizzazione. Dalla versione EJB 2.0 in poi, per assolvere a queste funzioni, è stato adottato JAAS (Java Autentication and Authorizazion Service), parte integrante di Java 2 SDK 1.4, ritenuto ormai maturo ed affidabile per la gestione dei delicati e critici processi relativi alla sicurezza.

JAAS può essere utilizzato in due modalità:

1. delegare la gestione di autenticazione e autorizzazione interamente al container. In questo caso colui che sviluppa l'applicativo potrebbe non avere competenze specifiche relative alla sicurezza;
2. utilizzare le interfacce di JAAS per poter sviluppare proprie servlet o client per poter scrivere sistemi di autenticazione personalizzati.

In entrambi i casi vengono gestite le informazioni relative all'utente (principal) e al profilo (role).

Dalla versione EJB 3.0 in poi la gestione dell'autorizzazione può essere implementata con la tecnica delle annotazioni. Per ogni metodo e/o classe di un EJB possono essere definiti i ruoli che hanno diritto ad accedere ad una determinata risorsa. Nel capitolo successivo sarà illustrato un esempio pratico di configurazione.

Capitolo 3 - Implementazione del programma

3.1 Introduzione

In questo terzo capitolo è presentato l'esempio pratico di migrazione del programma per la gestione del settore calcio giovanile da php a J2EE. Questa fase esprime il risultato ottenuto rispetto a quanto previsto e approfondito nei capitoli precedenti: l'esigenza era quella di sperimentare un'implementazione di un applicativo di carattere enterprise, sono stati approfonditi alcuni argomenti di carattere teorico ed infine, con l'implementazione pratica, si è ottenuto così il risultato finale.

3.2 L'Application Server

L'application server conforme alle specifiche J2EE, utilizzato in questo lavoro, è Jboss 5 versione community prodotto dalla RedHat.

Disponibile sul mercato anche in versione commerciale, per applicazioni *mission critical,* con supporto per piattaforme certificate compresa l'integrazione con altri prodotti come hybernate, Seam, jBPM e RichFaces, ha una comunità piuttosto attiva e non vi sono sostanziali differenze fra versione commerciale e community. Rispetto ad altre soluzioni open, come ad esempio GlassFish, dove l'acquisizione da parte di Oracle di Sun Microsystem ha determinato modifiche sostanziali di carattere strategico aziendale, con la conseguente incertezza sullo sviluppo dei prodotti Sun, RedHat rimane oggi l'azienda più importante nel mondo open source e quindi un riferimento significativo per questa tipologia di prodotti. Oltre a jBoss e Glassfish un'alternativa potrebbe essere Geronimo prodotto dalla Apache fondation, ma anche in questo caso all'interno di Apache negli ultimi tempi si sono verificate modifiche strutturali aziendali molto importanti, ad esempio IBM, forte sponsor del

progetto Geronimo ha deciso di sospendere il supporto al progetto Apache preferendo la collaborazione con Oracle.

3.3 DBMS

Come per la versione in php del programma, il dbms utilizzato è postgresql versione 8.4. un prodotto open source utilizzabile senza vincoli di licenza con la possibilità di poter acquisire la versione commerciale fornita dalla società EnterpriseDB. La scelta su questo dbms è legata alle esperienze avute in passato con questo prodotto e alle note vicende di acquisizione da parte di Oracle di MySql.

3.4 LDAP

Uno degli argomenti di discussione, che in passato sono stati oggetto di confronto con altri colleghi, riguarda la gestione delle utenze che nella stragrande maggioranza dei casi è implementata a livello applicativo e nello stesso database del programma. Personalmente ho sempre considerato questo approccio, sia in termini di sicurezza che di architettura nel suo insieme, estremamente critico. Nella versione del programma in php, le utenze erano gestite dal database, ma anche questa soluzione, se da un lato delegava al dbms la gestione dei processi di autenticazione e autorizzazione, dall'altro, presentava delle limitazioni. Ad ogni connessione di un utente il processo di accesso ai dati implicava la creazione di un socket fra il server web ed il DBMS. Questa soluzione non era in grado di scalare con un numero elevato di accessi in quanto, ad esempio, a fronte di qualche migliaio di utenti connessi contemporaneamente all'applicativo, il sistema operativo avrebbe potuto avere problemi di gestione delle connessioni di rete.

La gestione dell'autenticazione ed autorizzazione prevista dalle specifiche J2EE risolve questi problemi. Il container J2EE ha al suo interno i componenti per configurare l'accesso alle risorse, sia in termini di

autenticazione che autorizzazione, con il vantaggio di isolare questi processi dallo sviluppo della propria applicazione. La realizzazione di moduli per la gestione delle utenze si può avvalere di protocolli e soluzioni ormai consolidate nel tempo ed altamente affidabili. E' a questo proposito che è stata provata una soluzione che si basa su protocollo LDAP; utenze e ruoli sono gestiti dal server openldap.

La figura 3.1 illustra come, rispetto alla precedente versione del programma, il J2EE container isoli i processi di autenticazione e autorizzazione dall'applicativo e dal dbms.

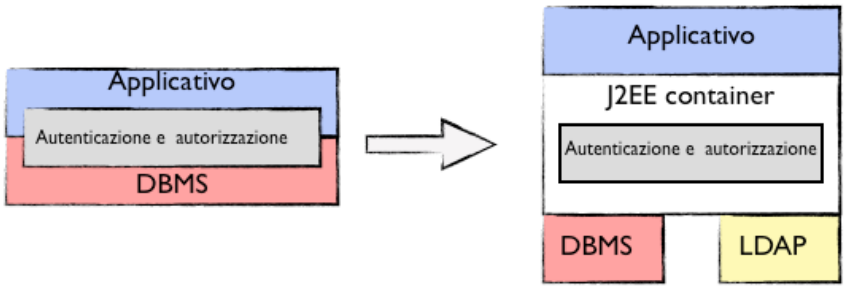


Figura 3.1 - Gestione dell'autenticazione ed autorizzazione

3.5 Installazione e configurazione dell'Application Server jBoss

L'istallazione dell'application server jBoss, versione 5.1.0, non ha richiesto particolari configurazioni, ci si è limitati a rispettare i prerequisiti previsti e ad utilizzare la configurazione di default. Il pacchetto è distribuito come file compresso, può essere installato sotto una propria cartella e può essere eseguito come utente standard. Visto le tecnologie che adotta, la sua struttura è piuttosto articolata e complessa e non è stata oggetto di approfondimenti in questo lavoro di tesi.

La configurazione di default è stata integrata con la definizione dei data-source e del contesto di sicurezza. L'accesso alle informazioni verso il database, da parte del codice applicativo, è stratificato a più livelli ognuno dei quali definito da specifici file di configurazione. La risorsa che permette di poter collegare jBoss ad un database prende il nome di data-source ed è definito da un file di configurazione in formato XML nel quale è possibile specificare i parametri necessari alla connessione come indirizzo ip e porta del dbms, il nome del database, il driver jdbc ecc.
Il listato seguente definisce il datasource per la connessione verso il database utilizzato dal programma.

Soccer-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>Soccer</jndi-name>
    <use-java-context>false</use-java-context>
    <connection-url>jdbc:postgresql://192.168.82.100:5432/soccer</connection-url>
    <driver-class>org.postgresql.Driver</driver-class>
    <user-name>soccer</user-name>
    <password>scp234!S</password>
    <min-pool-size>2</min-pool-size>
    <max-pool-size>4</max-pool-size>
  </local-tx-datasource>
</datasources>
```

L'application server svolge funzioni molto più articolate della semplice connessione al dbms; ad esempio, in base al carico verso il database, gestisce un pool di connessioni secondo i parametri min-pool-size e max-pool-size.

Per la parte relativa alla sicurezza, jBoss utilizza JAAS [Jam-Joh-2009] che può essere configurato tramite il file login-config.xml. E' interessante notare come siano disponibili vari livelli di astrazione, in modo da poter isolare le funzionalità da un livello all'altro e, allo stesso tempo, garantire un'elevata flessibilità di configurazione. Ad esempio, è possibile poter gestire le utenze e ruoli in file di testo, all'interno di un database e via LDAP senza la necessità di modifica alcuna nel codice dell'applicativo che accede alla risorsa. Questo dimostra come l'implementazione di applicativi in realtà complesse ed estremamente articolate, possano giovarsi di queste funzionalità al punto da poter accedere a sorgenti dati con estrema flessibilità. Quella seguente è la configurazione, lato application server, relativa alla sicurezza dell'applicativo. In questo caso, si possono, notare l'utilizzo di un server ldap e le specifiche parti dell'albero utilizzato per le utenze e per i profili. Anche in questo caso, con la tecnica delle definizioni dei prefissi e suffissi, può essere utilizzato un albero ldap arbitrario, con il risultato che a tutti i livelli viene garantita la massima flessibilità possibile.

login-config.xml

```
<application-policy name="ldapLogin">
      <authentication>
       <login-module flag="required"
code="org.jboss.security.auth.spi.LdapLoginModule">
          <module-option name="java.naming.factory.initial">
                                    com.sun.jndi.ldap.LdapCtxFactory</module-option>
          <module-option name="java.naming.provider.url">ldap://192.168.82.100:389/
                    </module-option>
          <module-option name="java.naming.security.authentication">simple
                    </module-option>
          <module-option name="principalDNPrefix">uid=</module-option>
          <module-option name="principalDNSuffix">,ou=Users,dc=gsbt,dc=it
```

```
                </module-option>
            <module-option name="rolesCtxDN">ou=Roles,dc=gsbt,dc=it</module-option>
             <module-option name="uidAttributeID">member</module-option>
             <module-option name="matchOnUserDN">true</module-option>
             <module-option name="roleAttributeID">cn</module-option>
              <module-option name="roleAttributeIsDN">false </module-option>
               <module-option name="allowEmptyPasswords">false</module-option>
        </login-module>
    </authentication>
</application-policy>
```

Nella figura 3.4 è illustrata la collocazione dei file soccer-ds.xml e login-conf.xml nel filesystem.

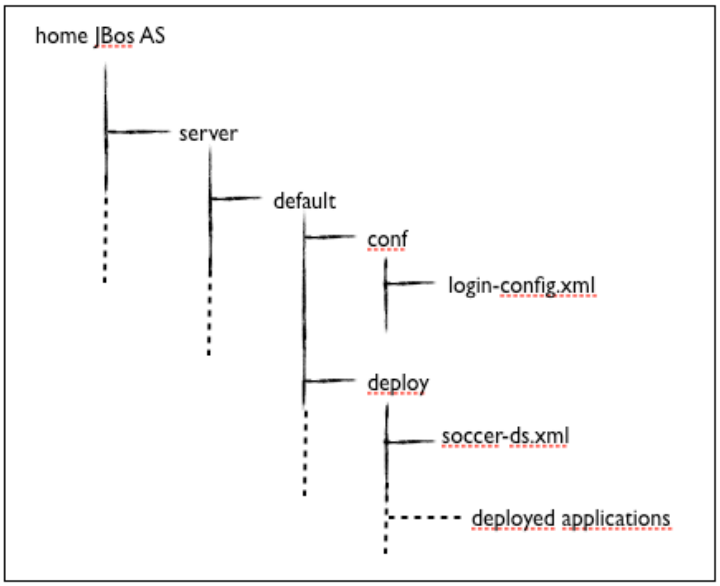


Figura 3.4 parte della struttura delle directory di jBoss

3.6 Ambiente di sviluppo

L'IDE utilizzato per lo sviluppo del programma è Eclipse JEE versione "Helios" con plugin LDAP e jBoss. Da un unico ambiente di sviluppo è stato possibile poter interagire sia con il server ldap che con l'application server; la figura 3.5 illustra l'organizzazione degli archivi utilizzati, per lo sviluppo del programma, di seguito elencati:

1. progetto per lo sviluppo delle servlet, pagine jsp, pagine html e librerie javascript (figura 3.5 riquadro "servlet");
2. progetto per l'implementazione degli EJB e Entity Bean (figura 3.5 riquadri EJB e Entity Bean);
3. progetto per l'aggregazione dei precedenti pacchetti in formato **E**nterprise Application **AR**chive .

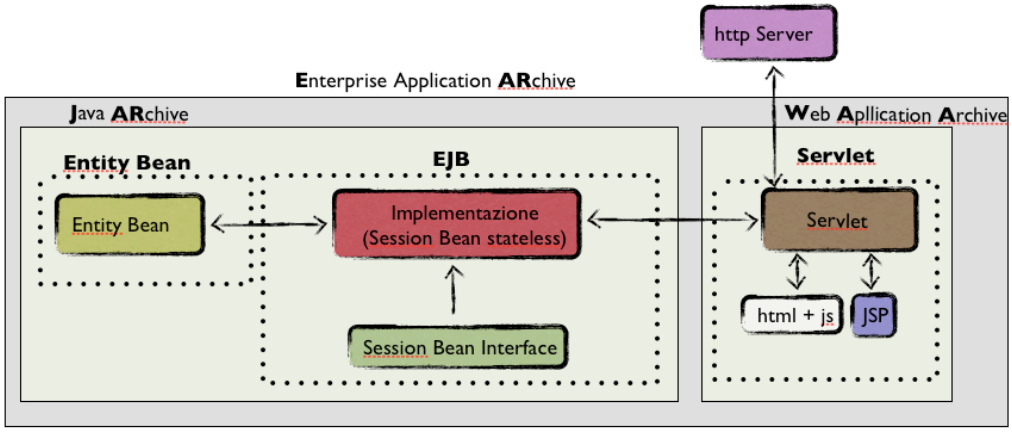


Figura 3.5 Organizzazione degli archivi

Le specifiche *Java Servlet* definiscono la struttura di un *Web Application Archive* suddivisa in una parte pubblica, accessibile cioè via URL, e ad una parte privata costituita da due sotto directory: META-INF che può contenere informazioni di *deployment* per la creazione di file war e WEB-INF (esempio in figura 2.3) così organizzata:

1. directory **classes**. Può contenere servlet, classi di utilità e JavaBean ;
2. direcotry **lib**. Può contenere librerie Java (.jar file) che sono disponibili solo all'*Application Web*;

3. directory **tags**. Può contenere librerie che possono essere richiamate da pagine JSP con la tecnica *tag library*;
4. file **web.xml** utilizzato dal servlet engine per l'esecuzione dell'applicazione. In questo file possono essere specificate tutta una serie di opzioni, come l'associazione fra servlet e URL (mapping), la definizione dei vincoli di sicurezza di alcune parti dell'application web , la configurazione di listner che possono essere associati a specifici eventi di un'applicazione ecc.

Come per i file war il formato JAR è un archivio compresso che può essere manipolato con l'utility jar parte di JDK e del tutto analogo al comando GNU tar. Viene utilizzato per la distribuzione di applicazioni java o di librerie e al suo interno, nella directory META-INF, è possibile definire informazioni per l'utilizzo del software. Ad esempio, nel caso degli Entity Bean il file persistence.xml definisce i contesti di persistenza dove ad ognuno dei quali sono associati alla sorgente dati e il tipo di ORM utilizzato.
Il listato seguente riporta il file persistence.xml del programma dal quale si può vedere l'associazione fra il data source Soccer definito in Soccer-ds.xml dell'application server e l'ORM Hibernate.

persistence.xml

```
<persistence>
   xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
   version="1.0">
 <persistence-unit name="PersistenceGiocatori">
   <jta-data-source>Soccer</jta-data-source>
    <properties>
      <property name="hibernate.dialect"
           value="org.hibernate.dialect.PostgreSQLDialect"/>
    </properties>
 </persistence-unit>
 ..............
```

```
  .............
</persistence>
```

Gli archivi di tipo EAR sono anch'essi in formato compresso e prevedono la possibilità di contenere moduli *.war, *.jar, *.rar e la directory META-INF che contiene informazioni sul deployment del pacchetto.

L'utilizzo di un ambiente di sviluppo come eclipse permette, in modo automatico, di gestire varie tipologie di progetti associati a specifiche tipologie di archivi.

3.7 Descrizione della struttura del programma

L'accesso ai contenuti dinamici è consentito grazie all'esecuzione di servlet che restituiscono informazioni in opportuni formati XML e Json utilizzati dalle librerie jquery per la loro visualizzazione e gestione degli eventi lato utente.

Le servlet sono lo strumento tramite il quale:

1. **gestire le richieste da parte dell'utenza**. Gli oggetti jQuery invocano le servlet con valori di input necessari per ottenere specifiche informazioni. Ad esempio, nel caso dell'oggetto jQgrid, per la visualizzazione dei dati, la servlet dovrà restituire in formato XML uno specifico numero di righe di una determinata pagina. Questo significa che in generale una tabella di n record viene suddivisa in tante pagine ognuna delle quali costituita da un certo numero di righe (figura 1.3). Oltre alla visualizzazione sono gestiti gli eventi di modifica, cancellazione, inserimento di un nuovo record, l'ordinamento della tabella e la sua stampa. L'interazione fra utente e applicativo implica l'attenta gestione dei valori delle variabili di input che potenzialmente potrebbero determinare seri problemi di sicurezza. Alcuni utenti potrebbero cercare di invocare le servlet con il passaggio di parametri tali da forzare i meccanismi di sicurezza del programma. A questo

scopo è stato utilizzato il validatore JSR 303 che sarà descritto nei paragrafi successivi;

2. **gestire l'accessibilità alle pagine web.** I contenuti dinamici dell'applicazione sono gestiti dalle servlet non vi sono quindi pagine JSP direttamente accessibili da parte dell'utente;
3. **gestire gli stati**. Lo stato della connessione viene gestito a livello servlet. Questo ha permesso di poter implementare la business logic con EJB session bean di tipo stateless estremamente efficienti e più semplici da gestire rispetto ai session bean di tipo statefull. In un contesto di carattere enterprise dove il numero di istanze EJB potrebbe essere molto elevato, ha fatto propendere per l'utilizzo di oggetti più semplici da gestire e meno onerosi in termini di risorse.

3.7.1 EJB ed Entity Bean

Gli EJB e gli Entity Bean sono gli oggetti utilizzati per l'implementazione della business-logic e per la persistenza dei dati. L'applicazione è stata suddivisa in moduli; per esempio gestione anagrafica, gestione tesseramento giocatori, gestione visite mediche ecc., dove ad ognuno di essi è stato associato lo sviluppo di un EJB ed Entity Bean. L'autorizzazione all'esecuzione dei metodi di un EJB è garantita da JAAS; in questo modo, sia la parte application web, che business logic sono protetti secondo regole prestabilite. Ad esempio, un utente con profilo "allenatore" non ha diritto ad apportare modifiche di nessun genere all'archivio anagrafico dei giocatori. L'inibizione dell'esecuzione di una determinata attività assegnata ad un profilo viene gestita dal server che è il responsabile primo della sicurezza del programma. Il lato client dell'applicativo si limita a visualizzare gli oggetti grafici che permettono di effettuare certe operazioni; in questo ambito non sono gestite operazioni di sicurezza.

3.7.2 Logger

Una delle parti fondamentali del programma è il logger che registra informazioni sull'attività dell'applicativo, sia per fini di debug e controllo in genere, sia per fini legati alla sicurezza. Ad esempio, le operazioni verso il database sono registrate come eventuali errori di validazione e integrità. L'implementazione del logger è piuttosto semplice, anche in questo caso viene utilizzato un pacchetto esterno denominato log4j, integrato con jBoss, che implementa le API per la gestione dei messaggi di log altamente configurabili.

3.7.3 Validator JSR 303

La validazione dei dati è una parte molto importante dell'applicativo in quanto evita la generazione di eccezioni sulle operazioni di I/O fra entità e database, con conseguenze significative sia sul buon funzionamento del programma che sulla sua sicurezza. Solitamente, l'implementazione della validazione è distribuita sui vari livelli applicativi, dal livello utente a quello di accesso ai dati, con il risultato che spesso la stessa logica è presente su diversi strati dell'applicativo con una elevata probabilità di errori e difficoltà di gestione e sviluppo.

Le API definite nel progetto JSR 303 permettono di gestire la validazione dei dati con un modello completamente diverso, estremamente efficiente e centralizzato. Tramite la tecnica delle annotazioni, per una classe, è possibile specificare vincoli di validazione su variabili, metodi e sull'intero oggetto. Nel caso della validazione dati relativa all'inserimento e/o modifica di nuovi record nel database, a livello di entity bean, sono stati specificati i vincoli di validazione; in questo modo, le servlet prima di chiamare i metodi della business logic per le operazioni CRUD invocano i metodi di JSR 303 per la validazione dei dati ed eventualmente restituire messaggi di errore al client. Il grande vantaggio di questo sistema è che le specifiche di validazione sono collocate in un solo punto e non distribuite ai vari livelli dell'applicativo.

3.7.8 Integrità

La gestione dell'autorizzazione ad eseguire una servlet e/o un modulo di una business logic, è associata al ruolo che ha un utente. Un utente con profilo "Presidente" potrà fare certe cose, un utente con profilo "Segreteria" potrà farne altre e così via.

Tutto ciò non è sufficiente per la gestione di più società, nel senso che un determinato userid di una certa società appartenente ad un determinato ruolo, non potrà accedere per nessun motivo a record di altre società. Nella gestione degli stati della connessione client-applicativo, al momento del login, vengono registrate nella variabile di sessione le seguenti informazioni:

1. nome e cognome;
2. ruolo;
3. nome società;
4. **id società**.

Prima dell'esecuzione di un metodo della business logic, la servlet controlla che i record acceduti tramite istruzioni sql non contengano riferimenti a società con id società diversi da quelli di appartenenza.

Per evitare infine che l'utente possa inserire nei parametri di input delle servlet invocate la richiesta di visualizzazione di campi riservati, tramite la proprietà riflessiva del linguaggio Java, è stata implementata una classe che permette di esporre soltanto alcuni campi di un'entità e non altri.

3.7.9 Gestione anagrafica giocatori

Illustrare tutto il codice del programma sarebbe troppo corposo ed inutilmente prolisso. Come esempio viene presentato il codice necessario all'implementazione della gestione anagrafica dei giocatori partendo dalla realizzazione dell'Entity Bean, per arrivare fino alla servlet richiamata dalle pagine web.

3.7.9.1 Entity Bean

L'implementazione dell'entity bean AnagraficaGiocatori si basa sull'utilizzo delle annotazioni con le quali il java bean è definito di tipo entità, è associato alla tabella dati anagrafica_giocatori dove sono specificati i vincoli di validazione sia a livello di variabile che a livello di metodo, ogni metodo get e set è associato ad un campo della tabella con la definizione del campo chiave (annotazione @Id).

AnagraficaGiocatori.java

```
package it.gsbt.Soccer.EJB.Entity;
import javax.persistence.*;
import java.io.Serializable;
import java.util.Date;
import javax.validation.constraints.*;
import org.hibernate.validator.constraints.NotEmpty;


@Entity
@Table(name="anagrafica_giocatori")
public class AnagraficaGiocatori implements Serializable {
private static final long serialVersionUID = 1L;
long id;
long idsocieta;

@NotEmpty(message="Il cognome non può essere nullo.")
String cognome;

@NotEmpty(message="Il nome  non può essere nullo.")
String nome;

Date datadinascita;

@NotEmpty(message="Il luogo di nascita  non può essere nullo.")
String luogodinascita;

@NotEmpty(message="Il codice fiscale  non può essere nullo.")
String codicefiscale;

@NotEmpty(message="La nazionalità non può essere nullo.")
String nazionalita;

@NotEmpty(message="La via  non può essere nullo.")
String via;

@NotEmpty(message="la città non può essere nullo.")
String citta;

@NotEmpty(message="Il CAP non può essere nullo.")
String cap;
```

```
@NotEmpty(message="Il provincia  non può essere nullo.")
  String provincia;

  @NotEmpty(message="Il telefono  non può essere nullo.")
  String telefono;

  String cellulare;
  String email;

  public AnagraficaGiocatori() {
                super();
                        }

  // Costruttore per operazioni di modifica record;
  public AnagraficaGiocatori(long id, String cognome, String nome,          Date datadinascita,String
luogodinascita,
            String codicefiscale, String nazionalita,String via,
            String citta, String cap, String provincia,
            String telefono,String cellulare,String email) {
  super();
  this.id=id;
  this.cognome = cognome;
  this.nome = nome;
  this.datadinascita = datadinascita;
  this.luogodinascita= luogodinascita;
  this.codicefiscale= codicefiscale;
  this.nazionalita= nazionalita;
  this.via= via;
  this.citta= citta;
  this.cap= cap;
  this.provincia= provincia;
  this.telefono= telefono;
  this.cellulare= cellulare;
  this.email= email;

  //entitylogger=new ApplicationLogger();

}
  // Cotruttore per insert record
  public AnagraficaGiocatori(String cognome, String nome, Date DataDiNascita,String
luogodinascita,
String codicefiscale,String nazionalita,
String via, String citta, String cap, String provincia,
                                        String telefono,String cellulare,String email) {
    super();
    this.cognome = cognome;
    this.nome = nome;
    this.datadinascita=DataDiNascita;
    this.luogodinascita=luogodinascita;
    this.codicefiscale=codicefiscale;
    this.nazionalita=nazionalita;
    this.via=via;
    this.citta=citta;
    this.cap=cap;
    this.provincia=provincia;
    this.telefono=telefono;
```

```
    this.cellulare=cellulare;
    this.email=email;

  }



  @Id
  @Column(name="id_giocatore")
  @SequenceGenerator(name="sequence",
sequenceName="anagrafica_giocatori_id_giocatore_seq")
  @GeneratedValue(generator = "sequence", strategy = GenerationType.SEQUENCE)
  public long getId() {
    return id;
  }
  public void setId(long id) {
    this.id = id;
  }

  @Column(name="id_societa")
  public long getIdsocieta() {
    return idsocieta;
  }
  public void setIdsocieta(long idSocieta) {
    this.idsocieta = idSocieta;
  }


  @Column(name="nome")
  @Pattern(regexp = "[a-zA-Zòàèù' ]*",message="Nome: formato dati non corretto.")
   @Size(max=32,message="Nome: lunghezza  max 32.")
  public String getNome() {
    return nome;
  }

  public void setNome(String nome) {
    this.nome = nome;
  }

  @Column(name="cognome")
  @Pattern(regexp = "[a-zA-Zòàèù' ]*",message="Cognome: formato dati non corretto.")
  //@NotNull(message="Cognome: non può essere nullo")
  @Size(max=32, message="Cognome: lunghezza max 32.")
  public String getCognome() {
                return cognome;
              }

  public void setCognome(String cognome) {
                this.cognome = cognome;
              }

  @Column(name="datadinascita")
  @NotNull(message="Data di Nascita: non può essere nulla")
  public Date getDatadinascita() {
                return datadinascita;
              }
```

```
public void setDatadinascita(Date datadinascita) {
            this.datadinascita = datadinascita;
          }

@Column(name="luogodinascita")
@Pattern(regexp = "[a-zA-Zòàèù' ]*",message="Luogo di nascita: formato dati non corretto.")
@NotNull(message="Luogo di nascita: non può essere nullo")
@Size(max=32, message="Luogo di nascita: lunghezza min 1 - max 32.")
public String getLuogodinascita() {
            return luogodinascita;
          }

public void setLuogodinascita(String luogodinascita) {
            this.luogodinascita = luogodinascita;
          }

@Column(name="codicefiscale")
@Pattern(regexp = "[a-zA-Z0-9]*",message="Codice fiscale:   formato dati non corretto.")
@Size(min=16,max=16, message="Codice Fiscale : lunghezza non corretta.")
public String getCodicefiscale() {
            return codicefiscale;
          }

public void setCodicefiscale(String codicefiscale) {
            this.codicefiscale = codicefiscale;
          }

@Column(name="nazionalita")
@Pattern(regexp = "[a-zA-Zòàèù' ]*",message="Nazionalità: formato dati non corretto.")
@Size(max=32,message="Nazionalità: lunghezza min 1 - max 32..")

public String getNazionalita() {
            return nazionalita;
          }

public void setNazionalita(String nazionalita) {
            this.nazionalita = nazionalita;
          }

@Column(name="via")
@Pattern(regexp = "[a-zA-Zòàèù\\.,', 0-9]*",message="Via: formato dati non corretto.")
//@NotNull (message="Via: non può esere nullo.")
@Size(max=32,message="Via: lunghezza min 1 - max 32.")

public String getVia() {
            return via;
          }

public void setVia(String via) {
            this.via = via;
          }

@Column(name="citta")
@Pattern(regexp = "[a-zA-Zòàèù' ]*",message="Città: formato dati non corretto.")
//@NotNull (message="Città: non può esere nullo.")
@Size(max=32,message="Città: lunghezza min 1 - max 32.")
public String getCitta() {
```

```
        return citta;
    }

public void setCitta(String citta) {
        this.citta = citta;
    }

@Column(name="cap")
@Pattern(regexp = "[0-9]*",message="CAP: formato dati non corretto.")
@Size(min=2,max=6,message="CAP: lunghezza min. 2, max. 6.")
@NotNull (message="CAP: non può esere nullo.")
public String getCap() {
        return cap;
    }

public void setCap(String cap) {
        this.cap = cap;
    }

@Column(name="provincia")
@Pattern(regexp = "[a-zA-Z]*",message="Provincia: formato dati non corretto.")
@Size(max=2,message="Provincia: devono essere specificati 2 caratteri.")
@NotNull (message="Provincia: non può esere nullo.")
public String getProvincia() {
        return provincia;
    }

public void setProvincia(String provincia) {
        this.provincia = provincia;
    }

@Column(name="telefono")
@Pattern(regexp = "[0-9 ]*",message="Telefono: formato dati non corretto.")
@Size(max=32,message="Telefono: lunghezza massima consentita 32 caratteri.")
public String getTelefono() {
        return telefono;
    }

public void setTelefono(String telefono) {
        this.telefono = telefono;
    }

@Column(name="cellulare")
@Pattern(regexp = "[0-9 ]*",message="Cellulare: formato dati non corretto.")
@Size(max=32,message="Cellulare: lunghezza massima consentita 32 caratteri.")

public String getCellulare() {
        return telefono;
    }

public void setCellulare(String cellulare) {
        this.cellulare = cellulare;
    }

 @Column(name="email")
 @Pattern(regexp = "[0-9A-Za-z.]*@[0-9A-Za-z.]*",message="Email: formato dati non corretto.")
 @Size(max=32,message="Email: lunghezza massima consentita 32 caratteri.")
```

```
 public String getEmail() {
              return email;
             }

public void setEmail(String email) {
              this.email = email;
             }

public String print()
{
             return "Anagrafica Giocatori: \n" +
             "Id:" + this.id + "\n" +
             "Id Societa:" + this.idsocieta + "\n" +
             "Cognome:" + this.cognome + "\n" +
             "Nome:" + this.nome + "\n" +
             "Data di nascita:" + this.datadinascita + "\n" +
             "Nato a:" + this.luogodinascita + "\n" +
             "Codice fiscale:" + this.codicefiscale + "\n" +
             "Nazionalità:" + this.nazionalita + "\n" +
             "Via:" + this.via + "\n" +
             "Ciità:" + this.citta + "\n" +
             "CAP:" + this.cap + "\n" +
             "Provincia:" + this.provincia + "\n" +
             "Telefono:" + this.telefono + "\n" +
             "Cellulare:" + this.cellulare + "\n" +
             "Email:" + this.email + "\n" ;
}
```

3.7.9.2 EJB Session Bean

La parte di business logic per la gestione anagrafica dei giocatori è implementata dal session bean CatalogoGiocatori. Nell'interfaccia oltre dichiarazione dei metodi, con la tecnica delle annotazioni, è specificato il tipo: remoto (@Remote) se session bean e servlet sono eseguiti in JVM diverse o locale (@Local) in caso contrario. L'interfaccia dichiara i metodi del bean necessari alle operazioni di inserimento, modifica, cancellazione ed interrogazione. Nella sua implementazione è specificato il contesto di persistenza, cioè a quale sorgente dati il bean fa riferimento (vedi esempio persistence.xml), il nome con il quale l'oggetto è registrato in JNDI, il dominio di sicurezza a cui è associato (vedi ***login-config.xml*** nel paragrafo 3.5) e i vincoli di esecuzione dei metodi.

CatalogoAnagraficaGiocatori.java (definizione interfaccia)

```
import it.gsbt.Soccer.EJB.Entity.*;
import javax.ejb.Remote;
import java.util.Collection;
import java.util.Date;

@Remote

/*
 * Interfaccia per l'accesso ai dati anagrafici dei giocatori.
 * L'EJB è di tipo staless e definito come remoto in JNDI.
 *
 * @Author Giovanni Sabatini
 *
 */
public interface CatalogoGiocatori {
  // Insert new record
  public AnagraficaGiocatori AddRecord(String Cognome,
            String Nome, Date DataDiNascita,
            String luogodinascita,String codicefiscale,
            String nazionalita, String via, String citta,
            String cap, String provincia,
String telefono,String cellulare,String email, String username);

  public AnagraficaGiocatori AddRecord(AnagraficaGiocatori ag, String username);

  // UpdateRecord Record

  public String UpdateRecord(AnagraficaGiocatori ag, String username);

  // Remove record
  public String RemoveRecord(long id, String username);


  // Query data
  public Collection <AnagraficaGiocatori> SelectRecords(String query, int start, int limit, String
username );

  public String RetrieveXMLDataQuery(String query, int Page, int NumeroRecordPerPagina, String
username );
  public String RetrieveJsonDataQuery(String query,String username );
  public Boolean IntegrityCheck(long idGiocatore, long idSocieta, String username);
}
```

CatalogoAnagraficaGiocatoriImpl.java (implementazione interfaccia)

```
package it.gsbt.Soccer.EJB.Stateless.Impl;

import it.gsbt.Soccer.EJB.Entity.*;
import it.gsbt.Soccer.EJB.Logger.*;
import it.gsbt.Soccer.EJB.Stateless.Interface.*;
```

```
import java.util.Collection;
import java.util.Date;
import java.util.Iterator;

import javax.ejb.Stateless;
import javax.ejb.TransactionManagement;
import javax.persistence.*;
import java.io.Serializable;
import org.jboss.ejb3.annotation.SecurityDomain;
import javax.annotation.security.*;

// registra in jndi come anagrafica
@Stateless(mappedName="anagrafica")
@SecurityDomain("ldapLogin")
@TransactionManagement(javax.ejb.TransactionManagementType.CONTAINER)

public class CatalogoGiocatoriImpl implements Serializable, CatalogoGiocatori {
        private static final long serialVersionUID = 1L;

@PersistenceContext(unitName="PersistenceGiocatori")
  protected  EntityManager em;
  protected AnagraficaGiocatori giocatore;
  protected Collection <AnagraficaGiocatori> listaGiocatori;
  ApplicationLogger log=new ApplicationLogger();


@RolesAllowed({"Admin","Segreteria"})
  public AnagraficaGiocatori AddRecord(String cognome, String nome, Date DataDiNascita,
                        tring luogodinascita,String codicefiscale,String nazionalita,
                        String via, String citta, String cap, String provincia,
                        String telefono,String cellulare,String email, String username)
{
    log.LogInfo(username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.AddRecord()");
        // Initialize the form
        giocatore = new AnagraficaGiocatori(nome, cognome, DataDiNascita,luogodinascita,
                codicefiscale, nazionalita,  via, citta, cap, provincia, telefono, cellulare, email);
        em.persist(giocatore);
        return giocatore;
}

 @RolesAllowed({"Admin","Segreteria"})
  public String UpdateRecord(AnagraficaGiocatori ag, String username)
  {
          log.LogInfo
        (username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.UpdateRecord()\n
START ENTITY DUMP \n"   + ag.print() + "END DUMP.");
          em.merge(ag);
          return "OK";
  }


  @RolesAllowed({"Admin","Segreteria"})
  public String RemoveRecord(long id, String username)
  {
          giocatore = em.find(AnagraficaGiocatori.class, id);
```

```
            log.LogInf0
(username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.RemoveRecord()\n START
ENTITY DUMP:\n" + giocatore.print() + "END DUMP.");
            em.remove(giocatore);
            return "OK";


 }



public Collection <AnagraficaGiocatori>SelectRecords(String query,
                                                int start, int limit, String username)
 {
            log.LogInfo (username,
           "gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.GetAllRecords() - Query -> "
           + query);

            log.LogInfo(username,
           "gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.GetAllRecords() - START=" + limit
+          "LIMIT=" + limit + ")");

            Query listaRecords=em.createQuery(query);
           if(start > 0)
                        listaRecords.setFirstResult(start);
             if(limit > 0)
                        listaRecords.setMaxResults(limit);
            return listaRecords.getResultList();
 }


public String RetrieveXMLDataQuery(String query, int Page, int NumeroRecordPerPagina, String
username )
{
           String XmlData="<?xml version='1.0' encoding='utf-8'?>\n";
           int nr;
           int start;


           int TotalPages;
           int TotalRecords;


           if(NumeroRecordPerPagina<1)
                        NumeroRecordPerPagina=1;
           if(Page<1)
                        Page=1;


           Query listaRecords=em.createQuery(query);
           log.LogInfo(username,
           "gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.RetriveXMLDataQuery(" +
           query + "," + Page + "," + NumeroRecordPerPagina );

           TotalRecords=listaRecords.getResultList().size();
           log.LogInfo(username,
```

```
            "gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.RetriveXMLDataQuery(Numero
Record="
+ TotalRecords);

            TotalPages=TotalRecords/NumeroRecordPerPagina + 1;

            start=Page*NumeroRecordPerPagina-NumeroRecordPerPagina;
            listaRecords.setFirstResult(start);
    listaRecords.setMaxResults(NumeroRecordPerPagina);

    nr=listaRecords.getResultList().size();
            log.LogInfo
(username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.RetriveXMLDataQuery(Numero
Record=" + nr);

    XmlData+="<rows>\n<page>" + Page +"</page>\n";
    XmlData+="<total>" + TotalPages + "</total>\n";
    XmlData+="<records>" + TotalRecords + "</records>\n ";
    for (Iterator iter = listaRecords.getResultList().iterator(); iter.hasNext();)
    {
            AnagraficaGiocatori element = (AnagraficaGiocatori) iter.next();
            XmlData+="<row id='" + element.getId() + "'>\n";
            XmlData+="<cell>" + element.getId() + "</cell>\n";
            XmlData+="<cell>" + element.getCognome() + "</cell>\n";
            XmlData+="<cell>" + element.getNome() + "</cell>\n";
            XmlData+="<cell>" + element.getDatadinascita() + "</cell>\n";
            XmlData+="<cell>" + element.getLuogodinascita() + "</cell>\n";
            XmlData+="<cell>" + element.getCodicefiscale() + "</cell>\n";
            XmlData+="<cell>" + element.getNazionalita() + "</cell>\n";
            XmlData+="<cell>" + element.getVia() + "</cell>\n";
            XmlData+="<cell>" + element.getCitta() + "</cell>\n";
            XmlData+="<cell>" + element.getCap() + "</cell>\n";
            XmlData+="<cell>" + element.getProvincia() + "</cell>\n";
            XmlData+="<cell>" + element.getTelefono() + "</cell>\n";
            XmlData+="<cell>" + element.getCellulare() + "</cell>\n";
            XmlData+="<cell>" + element.getEmail() + "</cell>\n";
            XmlData+="</row>\n\n";

    }
    XmlData+="</rows>\n";
                    log.LogInfo
(username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.RetriveXMLDataQuery - return");
            return XmlData;
}

public String RetrieveJsonDataQuery(String query,String username )
            {
                    int nr;
                    String JsonData="";
                    Query listaRecords=em.createQuery(query);
                    log.LogInfo
(username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.RetriveJsonDataQuery(" + query
+")" );
                    nr=listaRecords.getResultList().size();
                    log.LogInfo
(username,"it.gsbt.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.RetriveJsonDataQuery(Numero
Record=" + nr);
```

```
                        for (Iterator iter = listaRecords.getResultList().iterator(); iter.hasNext();)
                        {
                                AnagraficaGiocatori element = (AnagraficaGiocatori) iter.next();
                                JsonData+=element.getId() + ":";
                                JsonData+=element.getCognome() + " " +  element.getNome()  + ";";


                        }
                        log.LogInfo
(username,"it.gsbt.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.RetriveJsonDataQuery" +
JsonData);
                        if(JsonData.equals(""))
                                return "-:Nessun Nominativo";
                        else
                                return JsonData.substring(0,JsonData.length()-1);

        }

        public Boolean IntegrityCheck(long idGiocatore, long idSocieta, String username)
        {

                        Query listaRecords=em.createQuery("FROM AnagraficaGiocatori WHERE id=" +
idGiocatore + " AND idsocieta=" + idSocieta);
                        log.LogInfo
(username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.IntegrityChheck FROM
AnagraficaGiocatori WHERE id=" + idGiocatore + " AND idsocieta=" + idSocieta );
                if(listaRecords.getResultList().size() > 0)
                {
                        log.LogInfo
(username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.IntegrityCheck: OK for
idGiocatore=" + idGiocatore + " idsocieta=" + idSocieta );
                        return true;
                }
                else
                {
                        log.LogInfo
(username,"gsbt.it.Soccer.EJB.Stateless.Impl.AnagraficaGiocatori.IntegrityCheck: FAILED for
idGiocatore=" + idGiocatore + " idsocieta=" + idSocieta );
                        return false;
                }

        }
}
```

La gestione della persistenza delle entità è effettuata dall'Entity Manager (fig. 2.10) tramite le chiamate ai metodi dell'oggetto *em.* I metodi get e set dell'entity bean permettono di manipolare le informazioni degli oggetti istanziati in memoria. Nel caso di più oggetti dello stesso tipo, come ad esempio nel caso di operazioni di search nel database, gli oggetti vengono gestiti con la tecnica delle *Collection* [Bru-06].

3.7.9.4 Implementazione servlet

L'accesso ai dati è gestita da cinque servlet principali:

1. la servlet IndexAnagraficaGiocatori.java che è mappata in web.xml come /anagraficaGiocatori/index.html che è la pagina web tramite la quale l'utente può accedere alle gestione dell'anagrafica giocatori, è strutturata da alcuni parti statiche e da riferimenti a plugin Jquery già menzionati nei precedenti paragrafi;
2. ModificaGiocatori, utilizzata per le operazioni di inserimento, modifica e cancellazione dei record con verifiche preliminari di integrità dei dati;
3. AnagraficaGiocatori, utilizzata per le operazioni di visualizzazione;
4. Print, utilizzata per la gestione della stampa tramite la libraria fpdf;
5. main_menu.jsp per la visualizzazione delle voci di menu.

Tutte le servlet prima di eseguire qualsiasi tipo di operazione effettuano verifiche sullo stato della sessione in corso che è fondamentale per la gestione dello stato delle connessioni. In generale le variabili di sessione hanno un tempo di vita prestabilito che può essere impostato nella configurazione dell'application server. Potrebbe quindi verificarsi che, nonostante le operazioni di autenticazione e autorizzazione siano corrette, la sessione sia scaduta con l'impossibilità di effettuare operazioni sui dati. A questo scopo è stata implementata la classe CheckSession con i metodi per effettuare le verifiche necessarie sullo stato delle variabili di sessione.

IndexAnagraficaGiocatori.java

```
package it.gsbt.Soccer.Servlet;

import it.gsbt.Soccer.EJB.Entity.AnagraficaGiocatori;
import it.gsbt.Soccer.EJB.Logger.ApplicationLogger;
import it.gsbt.Soccer.EJB.Stateless.Interface.*;
import it.gsbt.Soccer.EJB.lib.CheckSession;

import java.security.Principal;
import java.util.*;
import java.io.*;
import javax.naming.*;
```

```
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.persistence.EntityManager;
import java.util.Enumeration;
import java.lang.reflect.*;



public class prova extends HttpServlet {
	private static final long serialVersionUID = 1L;


    /**
     * @see HttpServlet#HttpServlet()
     */
    public prova() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		// TODO Auto-generated method stub
		HttpSession session = request.getSession();
		session.setAttribute("XmlData","doGet Method");
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		// TODO Auto-generated method stub



		ApplicationLogger lg=new ApplicationLogger();

		String[] FieldsMasq={"idsocieta","class"};
		CheckSession  cks = new CheckSession
("it.gsbt.Soccer.EJB.Entity.AnagraficaGiocatori", FieldsMasq);




		response.setContentType("text/xml;charset=utf-8");
		PrintWriter out = response.getWriter();
```

```
                    String info;
                    // Log info
                    //ApplicationLogger lg=new ApplicationLogger();
                    // Principal Name
                    String PrincipalName;
                       // numero di pagina richiesta
                       int NumeroPagina=Integer.parseInt(request.getParameter("page"));
                       // numero di righe per pagina
                       int RighePerPagina=Integer.parseInt(request.getParameter("rows"));
                       // compo/i di ordinamento
                       String SIdx =  request.getParameter("sidx");
                       // tipo di ordinamente (cres. o decr.)
                       String Sort =request.getParameter("sord");



                       Principal userPrincipal = request.getUserPrincipal();

                       //
                                 // **************
                                 // verifica ed inpostazione della sessione
                          int rc =cks.SetSession(request, response, SIdx,Sort);
                                 if (rc == CheckSession.ERROR ||
                                           rc == CheckSession.EXPIRED ||
                                           rc == CheckSession.NEW )
                                 {
                                           RequestDispatcher dispatcher;
                                           lg.LogInfo("NULL","gsbt.it.Soccer.Servlet.prova.java - Check
Session -> Redirect to Login URL." );
                                                     //dispatcher = getServletContext
().getRequestDispatcher("/Login/index.html");
                                                     //dispatcher.forward(request,response);
                                                String contextPath= "http://localhost:8080/Soccer/";
                                                     response.sendRedirect(response.encodeRedirectURL
(contextPath + "/AnagraficaGiocatori"));

                                                     return;
                                           }
                                          // Fine verifica Sessione.
                                          // ************************



                         HttpSession session = request.getSession();
                         info="gsbt.it.Soccer.Servlet.prova.java: Welcome back - ID Session: " +
session.getId();

                          // Tempo di vita massimo di un sessione fra una richiesta del client e la
successiva.
                          // Questa impostazione sovrascrive quella sepcificata in web.xml.
                          //session.setMaxInactiveInterval(10);
                          //session.setAttribute("User", userPrincipal.getName());
                          //session.setAttribute("LastAccess", session.getLastAccessedTime() );
                          info+="\n Username:" + session.getAttribute("User") +
                "\n Nome:" + session.getAttribute("nome") +
             "\n Cognome: "  + session.getAttribute("cognome") +
```

```
        "\n idsocieta: "  + session.getAttribute("idsocieta") +
        "\n Societa: "  + session.getAttribute("societa") +
        "\n Profilo: "  + session.getAttribute("ruolo");

                        session.setAttribute("path", "AnagraficaGiocatori");
                        session.setAttribute("navigation", "Gestione Anagrafica Giocatori");




                        info += "\n Access Time: " + session.getLastAccessedTime() + "\n Creation
Time session: " + session.getCreationTime();

                        lg.LogInfo(userPrincipal.getName(), info);



                    Context ctx;
                            try {
                                    ctx = new InitialContext();
                                    CatalogoGiocatori giocatori=(CatalogoGiocatori) ctx.lookup
("anagrafica");
                                    out.println(giocatori.RetrieveXMLDataQuery("FROM
AnagraficaGiocatori c WHERE c.idsocieta=" +
                                                    session.getAttribute("idsocieta") +
                                                    session.getAttribute("orderby"),
NumeroPagina, RighePerPagina, userPrincipal.getName()));
                            } catch (NamingException e) {
                                    // TODO Auto-generated catch block
                                    e.printStackTrace();
                            }


        }

}
```

AnagraficaGiocatoriModifica.java

```
package it.gsbt.Soccer.Servlet;


import it.gsbt.Soccer.EJB.Entity.AnagraficaGiocatori;
import it.gsbt.Soccer.EJB.Entity.UserSocieta;
import it.gsbt.Soccer.EJB.Logger.ApplicationLogger;
import it.gsbt.Soccer.EJB.Stateless.Interface.CatalogoGiocatori;
import it.gsbt.Soccer.EJB.Stateless.Interface.CatalogoLogger;
import it.gsbt.Soccer.EJB.Stateless.Interface.CatalogoProfili;
import it.gsbt.Soccer.EJB.lib.*;

import java.io.*;
```

```
import java.security.Principal;
import java.sql.Timestamp;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Collection;
import java.util.Date;
import java.util.Iterator;
import java.util.Set;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.validation.*;


//import org.hibernate.validator.ClassValidator;
//import org.hibernate.validator.InvalidValue;
//import org.hibernate.validator.Length;
//import org.hibernate.validator.NotEmpty;
//import org.hibernate.validator;




/**
 * Servlet implementation class AnagraficaGiocaoriModifica
 */
public class AnagraficaGiocatoriModifica extends HttpServlet {

	private static final long serialVersionUID = 1L;



	String oper;
	int idsocieta;
	String cognome;
	String nome;
	Date datadinascita;
	String luogodinascita;
	String codicefiscale;
	String nazionalita;
	String via;
	String citta;
	String cap;
	String provincia;
	String telefono;
	String cellulare;
	String email;
```

```
/**
 * @see HttpServlet#HttpServlet()
 */
public AnagraficaGiocatoriModifica() {
    super();


    // TODO Auto-generated constructor stub

}

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                    // TODO Auto-generated method stub
                     ApplicationLogger lg=new ApplicationLogger();

        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */

        protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
        {

                    long idSocieta;
                    String info;
                    ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
            javax.validation.Validator validator = factory.getValidator();

            // La classe ag viene utilizzata per la validazione dati
            // (Vedi i constraints nell-Entity Bean Anagrafica Giocatori
            AnagraficaGiocatori ag = new AnagraficaGiocatori();

            ApplicationLogger lg=new ApplicationLogger();

            response.setContentType("text/html;charset=utf-8");
                    PrintWriter out = response.getWriter();

                    CheckSession  cks = new CheckSession();

                    // username utente autenticato
                    Principal userPrincipal = request.getUserPrincipal();
                    // **************
                    // verifica ed impostazione della sessione

                    lg.LogInfo("####: START ","gsbt.it.Soccer.Servlet.AnagraficGiocatoriModifica");
```

```
                        int rc =cks.SetSession(request, response);
                        if (rc == CheckSession.ERROR ||
                                        rc == CheckSession.EXPIRED ||
                                        rc == CheckSession.NEW )
                        {
                            RequestDispatcher dispatcher;
                                lg.LogInfo
("NULL","gsbt.it.Soccer.Servlet.AnagraficaGiocatoriModifica.java - Check Session -> Redirect to
Login URL." );
                                dispatcher = getServletContext().getRequestDispatcher("/Soccer/
Login/index.html");
                                dispatcher.forward(request,response);
                                //out.println("Attenzione la sessione è scaduta.");
                                lg.LogInfo("####: END
","gsbt.it.Soccer.Servlet.AnagraficGiocatoriModifica");
                                        return;
                                }
                         // Fine verifica Sessione.
                                // ************************


                        HttpSession session = request.getSession();
                        info="\n Username:" + session.getAttribute("User") +
            "\n Nome:" + session.getAttribute("nome") +
        "\n Cognome: "  + session.getAttribute("cognome") +
        "\n idsocieta: "  + session.getAttribute("idsocieta") +
        "\n Societa: "  + session.getAttribute("societa") +
        "\n Profilo: "  + session.getAttribute("ruolo")+
        "\n Access Time: " + session.getLastAccessedTime() +
        "\n Creation Time session: " + session.getCreationTime();

                        lg.LogInfo(userPrincipal.getName
(),"gsbt.it.Soccer.Servlet.AnagraficGiocatoriModifica - Ceck Session OK :\n" + info );


                if(request.getParameter("oper").equals("add") ||  request.getParameter("oper").equals
("edit"))
                {
                  // Tramite la classe CheckDate controllo la validita' della data
                  CheckDate date = new CheckDate();
                  date.StringToDate(request.getParameter("datadinascita"));
                  lg.LogInfo(userPrincipal.getName(),
"gsbt.it.Soccer.Servlet.AnagraficaGiocatoriModifica: START Validation Data");

                  ag.setIdsocieta((Long) session.getAttribute("idsocieta"));
                  ag.setNome(request.getParameter("nome"));
                  ag.setCognome(request.getParameter("cognome"));
                  ag.setDatadinascita(date.StringToDate(request.getParameter("datadinascita")));
                  if(request.getParameter("luogodinascita").length() ==0)
                      ag.setLuogodinascita(null);
                  else
                      ag.setLuogodinascita(request.getParameter("luogodinascita"));
                  ag.setCodicefiscale(request.getParameter("codicefiscale"));
                  ag.setNazionalita(request.getParameter("nazionalita"));
                  ag.setVia(request.getParameter("via"));
                  ag.setCitta(request.getParameter("citta"));
                  ag.setCap(request.getParameter("cap"));
```

```
                    ag.setProvincia(request.getParameter("provincia"));
                    ag.setTelefono(request.getParameter("telefono"));
                    ag.setCellulare(request.getParameter("cellulare"));
                    ag.setEmail(request.getParameter("email"));



                    //Set<ConstraintViolation<AnagraficaGiocatori>> costraintViolation =
validator.validateValue(AnagraficaGiocatori.class, "cognome",request.getParameter("cognome"));

                    // Verifico l'eventaule violazione dei constraints (validazione_
                    Set<ConstraintViolation<AnagraficaGiocatori>> costraintViolation =
validator.validate(ag);
                    if(costraintViolation.size() >= 1)
                    {

                        //rc+= costraintViolation.iterator().next().getMessage() + " ";

                        // Visualizzo il messaggio di errore (Vedi maschera di inserimento jQGrid)
                        out.println(costraintViolation.iterator().next().getMessage());
                        lg.LogInfo(userPrincipal.getName(),
"gsbt.it.Soccer.Servlet.AnagraficaGiocatoriModifica: " +   costraintViolation.iterator().next
().getMessage());


                        return;
                    }
                    lg.LogInfo(userPrincipal.getName(),
"gsbt.it.Soccer.Servlet.AnagraficaGiocatoriModifica: END Validation Data");
                    //out.println("OK");

                }




                     // Se l-operaiozne di Validazione e' terminata con successo allora verifico il tipo
di poerazione da effettuare
                     // con i dati: request.getParameter("oper")=="add" equivale all'operazione di
add record
                     if(request.getParameter("oper").equals("add"))
                     {
                              //Context ctx;
                                try{

                                   // Effettuo il lookup (JNDI) per istanziare l-ggetto registrato come
"anagrafica"
                                   // Che corrisposnde all-implementazione dell'EJB Staless
CatalogoGiocatori.
                                   // CatalogoGiocatori e' l-interfaccia alla quale viene effettuato l-
upcasting
                                   // dell-implementazione della stessa chiamata
CatalogoGiocatoriImpl
                            Context ctx=new InitialContext();
                            CatalogoGiocatori giocatori=(CatalogoGiocatori) ctx.lookup("anagrafica");
```

```
                                lg.LogInfo(userPrincipal.getName(),
"gsbt.it.Soccer.Servlet.AnagraficaGiocatoriModifica: Insert Record: " +
                                        ag.getCognome() + "," + ag.getNome() + ","  + ", " +
ag.getDatadinascita() +
                                        ", " + ag.getLuogodinascita() + "," + ag.getCodicefiscale() +
                                        ", " + ag.getCodicefiscale() + ", " + ag.getNazionalita() + ", " +
ag.getVia() + ", " +
                                        ag.getCitta() + ", " + ag.getCap() + ", " + ag.getProvincia() + ",
" + ag.getTelefono() +
                                        ", " + ag.getCellulare() + ", " + ag.getEmail());

                                giocatori.AddRecord(ag, userPrincipal.getName());
                               // out.println("OK");
                                    } catch(NamingException e)
                                    {
                                            session.setAttribute("ejb", e.getMessage());
                                    }


                }

                        // Update Record
                        if(request.getParameter("oper").equals("edit"))
                        {

                                //Context ctx;
                                  try{

                                    // Effettuo il lookup (JNDI) per istanziare l-ggetto registrato come
"anagrafica"
                                    // Che corrisposnde all-implementazione dell'EJB Staless
CatalogoGiocatori.
                                    // CatalogoGiocatori e' l-interfaccia alla quale viene effettuato l-
upcasting
                                    // dell-implementazione della stessa chiamata
CatalogoGiocatoriImpl
                                Context ctx=new InitialContext();
                                CatalogoGiocatori giocatori=(CatalogoGiocatori) ctx.lookup("anagrafica");

                                if(giocatori.IntegrityCheck(Long.parseLong(request.getParameter("id")),
(Long) session.getAttribute("idsocieta"), userPrincipal.getName()))
                                {
                                  ag.setId(Integer.parseInt(request.getParameter("id")));

                                  lg.LogInfo(userPrincipal.getName(),
"gsbt.it.Soccer.Servlet.AnagraficaGiocatoriModifica: Update Record: " +
                                        ag.getId() + ", " + ag.getCognome() + "," + ag.getNome() + ","
+ ", " + ag.getDatadinascita() +
                                        ", " + ag.getLuogodinascita() + "," + ag.getCodicefiscale() +
                                        ", " + ag.getCodicefiscale() + ", " + ag.getNazionalita() + ", " +
ag.getVia() + ", " +
                                        ag.getCitta() + ", " + ag.getCap() + ", " + ag.getProvincia() + ",
" + ag.getTelefono() +
```

```
                                        ", " + ag.getCellulare() + ", " + ag.getEmail
());
                                giocatori.UpdateRecord(ag, userPrincipal.getName());
                                 //out.println("OK");
                                      }
                                   } catch(NamingException e1)
                                   {

                                        session.setAttribute("ejb", e1.getMessage());
                                   }

                }


                        // Delete Record
                        if(request.getParameter("oper").equals("del"))
                        {
                                //Context ctx;
                                  try{

                                     // Effettuo il lookup (JNDI) per istanziare l-ggetto registrato come
"anagrafica"
                                     // Che corrisposnde all-implementazione dell'EJB Staless
CatalogoGiocatori.
                                     // CatalogoGiocatori e' l-interfaccia alla quale viene effettuato l-
upcasting
                                     // dell-implementazione della stessa chiamata
CatalogoGiocatoriImpl
                                Context ctx=new InitialContext();
                                CatalogoGiocatori giocatori=(CatalogoGiocatori) ctx.lookup("anagrafica");

                                if(giocatori.IntegrityCheck(Long.parseLong(request.getParameter("id")),
(Long) session.getAttribute("idsocieta"), userPrincipal.getName()))
                                        {
                                 lg.LogInfo(userPrincipal.getName(),
"gsbt.it.Soccer.Servlet.AnagraficaGiocatoriModifica: Delete Record con ID= " +
                                        Integer.parseInt(request.getParameter("id")));
                                  giocatori.RemoveRecord(Integer.parseInt(request.getParameter("id")),
userPrincipal.getName());
                                        }
                             //out.println("OK");
                                   } catch(NamingException e1)
                                   {

                                        session.setAttribute("ejb", e1.getMessage());
                                   }
                }
                        out.println("OK");
                        lg.LogInfo("####: END ","gsbt.it.Soccer.Servlet.AnagraficGiocatoriModifica");

           }
}
```

Print.java

```
public class PrintPDF extends HttpServlet {
	private static final long serialVersionUID = 1L;


   /**
    * @see HttpServlet#HttpServlet()
    */
   public PrintPDF() {
      super();
      // TODO Auto-generated constructor stub
   }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		// TODO Auto-generated method stub
		doPost(request, response);
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
	{
		// TODO Auto-generated method stub


		Collection<AnagraficaGiocatori> lista;
		ApplicationLogger lg=new ApplicationLogger();

		String[] FieldsMasq={"idsocieta","class"};
		CheckSession  cks = new CheckSession
("gsbt.it.Soccer.EJB.Entity.AnagraficaGiocatori", FieldsMasq);

		response.setContentType("application/pdf;charset=utf-8");
		PrintWriter out = response.getWriter();
		Principal userPrincipal = request.getUserPrincipal();

			// verifica ed impostazione della sessione
		    int rc =cks.CheckSessionPrint(request, response,
"gsbt.it.Soccer.EJB.Entity.AnagraficaGiocatori");
			if (rc == CheckSession.ERROR ||
				rc == CheckSession.EXPIRED ||
				rc == CheckSession.NEW )
			{
				 RequestDispatcher dispatcher;
				 lg.LogInfo("NULL","gsbt.it.Soccer.Servlet.prova.java - Check
Session -> Redirect to Login URL." );
					//dispatcher = getServletContext
().getRequestDispatcher("/Login/index.html");
					//dispatcher.forward(request,response);
				String contextPath= "http://localhost:8080/Soccer/";
```

```
                                        response.sendRedirect(response.encodeRedirectURL
(contextPath + "/AnagraficaGiocatori"));

                                            return;
                                    }
                                    // Fine verifica Sessione.
                                    // ************************

                                    HttpSession session = request.getSession();
                                    //lg.LogInfo(Principal,"gsbt.it.Soccer.Servlet.Print.java: Welcome back
- ID Session: " + session.getId());

                                    // Se il metodo CehckSessionPrint restituisce ENTITYBEANMATCH
allora c'e' corrispondenza fra il nome
                                    // dell'EntityBean passato per argomento con il nome dell'entitybean
                                    if(rc== CheckSession.ENTITYBEANMATCH)
                                    {
                                            // info="gsbt.it.Soccer.Servlet.Print.java: Welcome back - ID
Session: " + session.getId();
                                            Context ctx;
                                            try {
                                                    ctx = new InitialContext();
                                                    CatalogoGiocatori giocatori=(CatalogoGiocatori)
ctx.lookup("anagrafica");
                                                    lista = giocatori.SelectRecords("FROM
AnagraficaGiocatori c WHERE c.idsocieta=" +


            session.getAttribute("idsocieta") +


            session.getAttribute("orderby"),

                                                                                    0, 0,
userPrincipal.getName() );


                                                    pdf doc = new pdf(lista);

                                                    out.println( doc.output());
                                                    lg.LogInfo("NULL","STAMPA Dociumento PDF" );




                                            } catch (NamingException e) {

                    e.printStackTrace();
                                                    }


                                    }



            }
```

```
}
```

main_menu.jsp

```
<%@ page language="java" contentType="text/javascript; charset=UTF-8"%>
[
	{
		"text": "1. Anagrafica Giocatori",
		<% if(session.getAttribute("path").equals("AnagraficaGiocatori"))
			out.print("\"expanded\": true,");
		  else
			out.print("\"expanded\": false,");

		%>

		"children":
		[
			{

				"text": "<a href=\"/Soccer/AnagraficaGiocatori/\">1.1
Inserimento-Modifica-Canc.</a>"
			},
			{
				"text": "1.2 Stampa in ordine Alf."
			}
		]
	},

	<% if (session.getAttribute("ruolo").equals("Admin") || session.getAttribute("ruolo").equals
("Segreteria"))
	{ %>
	{
		"text": "2. Visite Mediche",
			"expanded": false,

		"children":
		[
			{
				"text":"2.1 Inserimento-Modifica-Canc."
			},
			{
				"text": "2.2 Stampa in ordine Alf."
			},
			{
				"text": "2.2 Stampa per anno di nascita"
			}
		]
	},
	<%}%>

	{
		"text": "3. Tesseramento",

			"expanded": false,
```

```
                "children":
                [
                        {
                                "text":"3.1 Inserimento-Modifica-Canc."
                        },
                        {
                                "text": "3.2 Stampa in ordine Alf."
                        },
                        {
                                "text": "3.2 Stampa per anno di nascita"
                        }
                ]
        },
        {
                "text": "4. LogOut",

                        "expanded": false,

                "children":
                [
                        {
                                "text": "<a href=\"/Soccer/LogOut\">4.1 LogOut</a>"
                        }
                ]
        }


]
```

La figura 3.6 illustra lo schema generale relativo alla gestione dell'anagrafica giocatori compresi i package in cui le varie componenti sono state sviluppate.

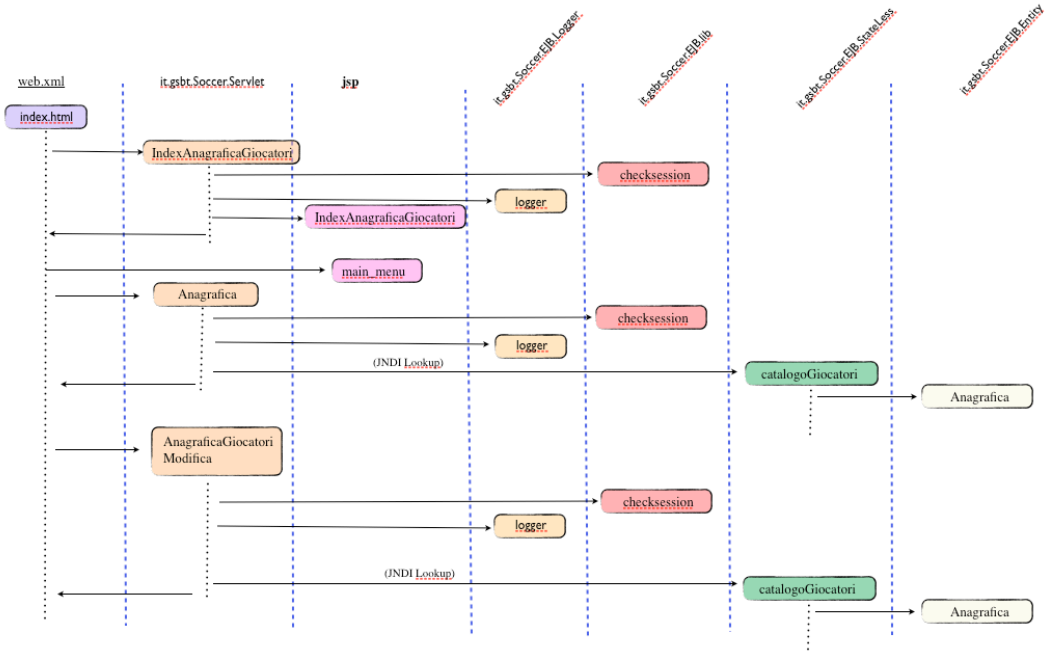


Figura 3.6 Schema generale relativo alla gestione anagrafica giocatori

3.7.9.5 Albero LDAP

L'albero ldap rappresentato in figura 3.6 illustra la struttura delle informazioni utilizzate dall'application server per i servizi di autenticazione ed autorizzazione secondo i criteri definiti nel file login-conf.xml.

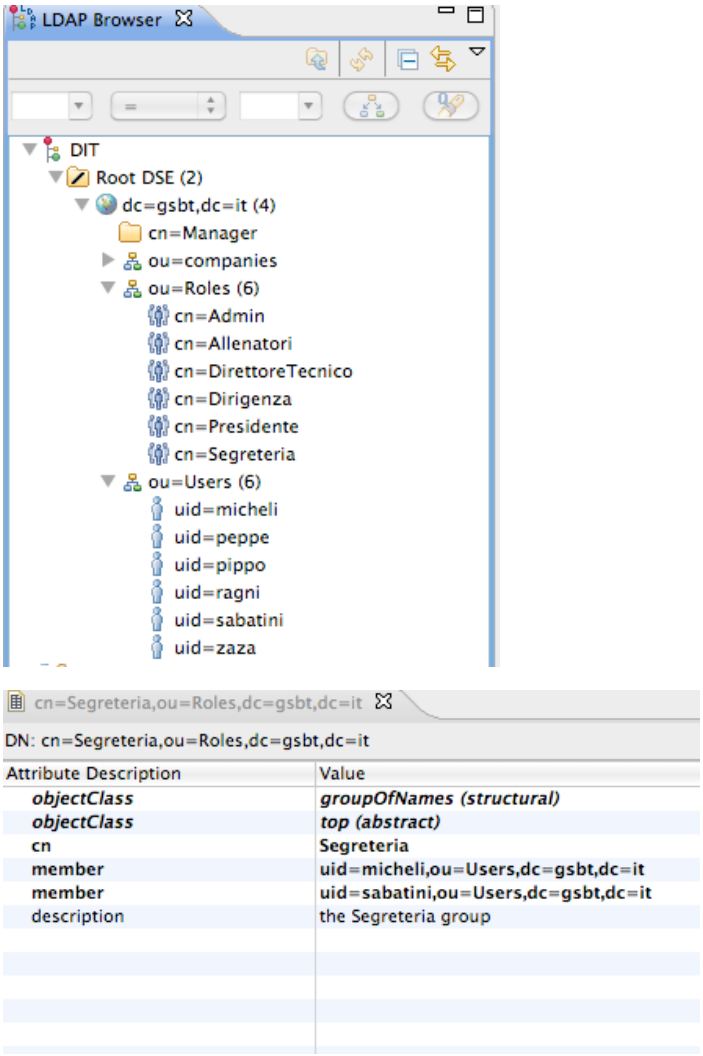


Figura 3.7 Struttura albero ldap

Ad esempio l'utente "sabatini" appartiene al ruolo "Segreteria" e quindi potrà svolgere operazioni autorizzate per quel profilo.

Conclusioni

Con questo lavoro di tesi sono stati approfonditi gli aspetti fondamentali dell'infrastruttura di base delle specifiche J2EE. Partendo da un esempio pratico relativo al programma di gestione del settore calcio giovanile, precedentemente sviluppato con altre tecnologie, è stato effettuato lo studio della sua migrazione in un contesto enterprise. Questo ha permesso di poter sperimentare gli effettivi vantaggi offerti dalla tecnologia J2EE: sviluppo multilivello del codice, separazione fra la logica dell'applicativo con funzionalità di carattere generale come autenticazione, validazione, autorizzazione, ecc. e l'utilizzo di tecnologie complesse consolidate ed affidabili. Decidere di sviluppare un applicativo con questa architettura, implica una maggiore competenza rispetto a quanto richiesto per altre soluzioni. L'approccio necessario alla realizzazione di una applicazione in J2EE è sicuramente più di carattere ingegneristico, rispetto ad esempio a soluzioni senza impiego di application server. Prima quindi di poter iniziare a produrre codice ed implementare l'applicativo, è stato indispensabile studiare l'architettura J2EE, per individuare le parti fondamentali sulle quali basare lo sviluppo del programma e affinare e approfondire le competenze in step successivi. A questo proposito si ricorda che tutta la parte relativa alla presentazione dei dati e alla gestione degli eventi utente, è stata volutamente implementata con delle servlet, realizzate senza l'ausilio di tecnologie come JSF o SEAM. L'obiettivo infatti era quello di prendere visione del funzionamento del sistema nelle sue parti fondamentali di basso livello. Lo sviluppo futuro o meglio la versione successiva del programma adotterà JSF e SEAM, ma con basi solide di consapevolezza e conoscenza di una tecnologia così complessa ed articolata. Detto questo, si può affermare che vi sono innumerevoli vantaggi insiti nelle specifiche J2EE rispetto a soluzioni senza l'impiego di un middle-wear e quindi apparentemente più produttive (il programmatore

inizia subito a scrivere il codice). La complessità di soluzioni conformi allo standard J2EE è sicuramente notevole e da un certo punto di vista può essere visto come un aspetto negativo. E' ovvio che l'impiego di questa tecnologia va valutata in funzione delle caratteristiche del servizio o procedure che si vanno ad implementare. L'aspetto negativo, se tale lo vogliamo definire, è che, a volte, ci si trova difronte a livelli così stratificati e articolati che si ha l'impressione di non riuscire ad individuare un quadro chiaro di ciò che si sta approfondendo, ecco perchè si è cercato di individuare quelle parti fondamentali necessarie e sufficienti per poter realizzare un'implementazione di base del programma.

In fase di analisi, prima di decidere se impiegare questa tecnologia per lo sviluppo di un'applicazione, andrebbero valutati i seguenti punti:

1. **bacino di utenza**. Se il bacino di utenza è considerevole, la gestione delle risorse diventa fondamentale. Poter sviluppare la propria applicazione in un'architettura che, per sua natura, è in grado di svolgere funzioni fondamentali come load balancing e alta affidabilità , senza dover modificare il codice della propria applicazione, dimostra come in alcuni contesti l'adozione di un application server J2EE sia una scelta necessaria. C'è da precisare che questa non è l'unica soluzione disponibile, vi sono altre tecnologie che possono offrire soluzioni alternative a J2EE, come ad esempio Microsoft.NET;
2. **manutenzione**. La modularità dell'architettura J2EE facilità lo sviluppo e la manutenzione dell'applicativo;
3. **portabilità**. La possibilità di poter migrare la propria applicazione da un application server ad un altro su diverse piattaforme, dà un valore aggiunto al prodotto che, in termini commerciali, potrebbe rivelarsi, dal punto di vista strategico, di fondamentale importanza;
4. **interfaccia a sistemi esterni**. Con la tecnica dei web services è possibile esporre alcune funzionalità del programma verso altri applicativi, con relativa semplicità

5. **accesso ai dati**. L'application server è in grado di gestire l'accesso ai dati aziendali in modo coerente e affidabile, anche se tutto ciò potrebbe portare a criticità sulle performance dell'applicazione.

Sviluppare un applicativo in un contesto enterprise porta ad identificare una serie di professionalità ben specifiche per il suo sviluppo del software, per la gestione dell'application server e l'ottimizzazione delle problematiche di sincronizzazione fra application server e database.

Questo lavoro di tesi, secondo me, è stato il primo passo per poter acquisire le basi fondamentali per lo sviluppo di applicazioni complesse di carattere enterprise su tecnologia J2EE. Mia ntenzione è quella di poter continuare ad approfondire queste tematiche, al fine di allargare le mie conoscenze e comptetenze professionali e, nello stesso tempo, do essere di stimolo anche per altri interessati a questa architettura.

Bibliografia

[Res-09] John Resig, *jQuery CookBook*, O'Reilly, 2009

[Bru-06] Bruce Eckel's, *Thinking in Java 4th Edition*, Prentice Hall, 2005

[Chop-Li-Gen-07] Vivek Chopra, Sing Li, Jeff Genender, Professional *Apache Tomcat 6*, Wrok

[Bas-Sie-Bat-2008] Bryan Basham, Kathy Sierra, Bert Bates, *Head First Servlets & JSP*, O'Reilly, 2008

[Srig-Bro-Silv-2006] Rima Patel Sgriganesh, Gerald Brose, Micah Silverman, *Mastering Enterprise JavaBeans 3.0 4th Edition, Wiley Plubblishing, 2006*

[Jam-Joh-2009] Javid Jamae, Peter Johonson , *Jboss in Action* , Manning, 2009