UNIVERSITÀ DEGLI STUDI DI MODENA
E REGGIO EMILIA

Facoltà di Ingegneria – Sede di Modena

Corso di Laurea in Ingegneria Informatica

Portale Web di Facoltà: progetto e implementazione di homepage dei docenti mediante architettura J2EE

Relatore

Chiar.mo Prof.

Sonia Bergamaschi

Correlatore

Ing. Maurizio Vincini

Tesi di Laurea di

Andrea Cervellati

Controrelatore

Chiar.mo Prof.

Flavio Bonfatti

Anno Accademico 2000-2001

Ringraziamenti

Desidero ringraziare la prof. Sonia Bergamaschi per avermi dato la possibilità di sviluppare questa tesi e per la disponibilità e la fiducia dimostrata.

Un ringraziamento anche all'ing. Maurizio Vincini e all'ing. Francesco Guerra per l'aiuto fornito durante lo sviluppo e la stesura della tesi.

Ringrazio la mia famiglia, e in particolar modo mia madre, per il calore e l'affetto sempre dimostrato.

Un ringraziamento speciale a Emanuela, per essermi stata così vicina e avermi sopportato in questo periodo così intenso.

Indice

Indice delle figure

Introduzione

Lo sviluppo di questa tesi si è articolato in due parti principali: una prima parte di riprogettazione del database attualmente presente per la gestione del portale Web di Facoltà, e una seconda parte di progettazione e implementazione di una particolare applicazione Web con pagine costruite dinamicamente su tale database.

Per quanto riguarda la prima parte, alla quale è dedicato il primo capitolo, ci si è occupati essenzialmente di eliminare alcune problematiche progettuali riscontrate sull'attuale database e prevedere l'integrazione in un'unica base di dati sia delle informazioni relative all'organizzazione didattica che di quelle riguardanti la Guida dello Studente.

La seconda parte, di carattere più applicativo, si riferisce alla realizzazione delle homepage dei docenti del sito della Facoltà di Ingegneria di Modena e Reggio Emilia. Attraverso la propria homepage il docente ha a disposizione un mezzo potente e comodo per far comparire on-line i propri dati personali, e pubblicare materiale di vario genere, come ad esempio il materiale didattico di ciascun insegnamento da lui tenuto.

Capitolo 1

Progetto e implementazione del database

Si prende in esame la riprogettazione del database del portale Web di Facoltà, partendo dalla situazione come era in precedenza, mettendo in risalto le problematiche affrontate e le soluzioni proposte, giungendo poi alla stesura di un nuovo schema relazionale.

1.1 Brevi richiami alla Teoria della Normalizzazione

Viene fatto un breve richiamo alla Teoria della Normalizzazione[NORM] di cui si è fatto uso in questa fase. Viene sottintesa la definizione di *schema di relazione*, *chiave* e *attributo primo*. Si riportano invece le definizioni di *dipendenza funzionale*, e di *dipendenza completa*.

Dato uno schema R(T) , una *dipendenza funzionale (FD)* su R è un vincolo di integrità $Y \tau Z$ essendo $Y,Z \subseteq T$.

Dato uno schema (R(T),F), un attributo $A \in T$ e un insieme di attributi $Y \subseteq T$, si dice che A *dipende completamente* da Y se $Y \tau A \in F$ e non esiste nessun sottoinsieme proprio Z di Y ($Z \subset Y$) tale che $Z \tau A$.

Ora si richiamano le definizioni delle varie forme normali che sono alla base della teoria della normalizzazione e consentono di stabilire la “qualità” di un progetto

relazionale. Uno schema relazionale che violi le forme normali presenta infatti delle ridondanze nei dati o delle anomalie di aggiornamento più o meno gravi a seconda del tipo di forma violata.

Le principali forme normali sono quattro:

1) Prima forma normale (1NF).

Uno schema R(T) è in *1NF* se e solo se i valori di tutti i domini degli attributi $A \in T$ sono *atomici*, cioè ogni attributo può assumere un valore elementare. Uno schema che viola la 1NF contiene ad esempio un attributo *set* che può assumere valore complesso, quale ad esempio una tupla. Nel modello relazionale si assume implicitamente che tutti gli attributi siano atomici e quindi sia soddisfatta la 1NF.

2) *Seconda forma normale (2NF)*.

Uno schema (R(T),F) è in *2NF* se e solo se ogni attributo non primo $A \in T$ dipende completamente da ognuna delle chiavi di R.

Uno schema che viola la 2NF si ha quando un attributo che non fa parte di una chiave (non primo) dipende funzionalmente solo da una parte di una chiave: es. R(<u>A,B</u>,C,D) con $B \tau D$.

3) *Terza forma normale (3NF)*.

Uno schema (R(T),F) è in *3NF* se e solo se per ogni dipendenza funzionale non banale $X \tau A \in F$, o X è una superchiave oppure A è primo.

Uno schema che viola la 3NF , ma non la 2NF si ha quando esiste almeno una dipendenza funzionale non banale nella quale il determinante non è superchiave e il determinato è non primo (cioè non appartiene a nessuna chiave): es. R(<u>A</u>,B,C,D) con $C \tau D$

Uno schema in 3NF pur non essendo esente da problemi di ridondanza e anomalie di aggiornamento, presenta comunque un buon livello di qualità progettuale.

1.2 Situazione precedente

Il primo passo che deve essere compiuto da un progettista quando si approccia al problema è quello di raccogliere tutte le informazioni relative alla realtà da modellare. Questa fase è detta *Specifica dei requisiti* ed è fondamentale per capire ciò che deve essere rappresentato nel database e quali sono i vincoli da rispettare.

La situazione ideale è quella in cui si riesce a racchiudere nelle specifiche tutte le possibili modifiche a cui può essere soggetta la realtà d'interesse a causa di eventi futuri che possono verificarsi. Ciò ovviamente non è semplice e alle volte risulta praticamente impossibile.

Da quando è partito il progetto del portale Web ad oggi, sono avvenuti notevoli cambiamenti, primo tra tutti il cambiamento dal vecchio al nuovo ordinamento didattico. Si è reso quindi necessario modificare il database in modo da comprendere i nuovi requisiti, tra i quali ad esempio l'introduzione dei crediti formativi che ciascun insegnamento fornisce e la riorganizzazione dei corsi in Settori Scientifico Disciplinari.

Come spesso accade nel dover aggiungere nuovi requisiti dei quali non si era tenuto conto in fase di progettazione, ad un database già esistente, possono verificarsi situazioni in cui non si riesce ad evitare l'introduzione di violazioni di una forma normale se non riprogettandolo ex novo. Solitamente però un progetto già operativo non può fermarsi per tempi troppo lunghi, come quelli necessari per una rielaborazione del progetto stesso; si preferisce quindi aggiungere i nuovi requisiti anche se questo può temporaneamente introdurre violazioni della Terza Forma Normale, per avere così il tempo per la riprogettazione, senza precluderne l'operatività.

La situazione immediatamente precedente lo sviluppo di questa tesi prevedeva una modellazione della realtà d'interesse attraverso due database relazionali distinti realizzati su due diversi DBMS.

Il primo database, implementato su SQL Server , conteneva le informazioni riguardanti l'organizzazione didattica di Facoltà, sia derivanti da direttive specifiche, come il Regolamento di Facoltà (RdF), il Regolamento di Corso di Laurea (RdCL) e il Manifesto degli studi, sia derivanti dall'organizzazione interna del personale accademico. Questa base di dati forniva la sorgente di informazioni cui si appoggiava la applicazione Web per la costruzione delle pagine dinamiche del sito della Facoltà di Ingegneria di Modena e Reggio Emilia.

Il secondo database, implementato su MS Access, conteneva dati che sostanzialmente erano utilizzati come sorgente di informazioni di riferimento per la stesura della Guida dello studente. In questa guida, che viene redatta in ciascun anno accademico, devono comparire una serie di informazioni riguardanti ogni facoltà, come ad esempio una breve presentazione, l'indicazione delle biblioteche, del preside e i vari delegati, e l'elenco di tutti i corsi di studio attivati nell'anno accademico di riferimento. Per ciascun corso devono poi essere riportati tutti gli insegnamenti con l'indicazione dell'anno di corso e del corrispondente numero di crediti formativi forniti.

Entrambi questi database sono stati oggetto delle modifiche necessarie all'introduzione dei requisiti dettati dal nuovo ordinamento didattico, mantenendo ovviamente anche quelli dell'ordinamento precedente almeno per un certo periodo di tempo transitorio necessario affinchè il nuovo arrivi a regime.

Il primo importante obiettivo che ci si è proposti è stato dunque quello di rielaborare il progetto esistente in modo da renderlo in Terza Forma Normale.

E' apparsa poi subito evidente una cruciale problematica dovuta all'esistenza di due sorgenti dati distinte contenenti in parte le stesse informazioni. L'elenco dei corsi attivati e dei relativi insegnamenti, ad esempio, è riportato anno per anno dal Manifesto degli studi di una certa facoltà, perciò doveva essere presente sul primo database, ma, dovendo comparire anche nella Guida dello studente, doveva essere presente anche nel secondo. Oltre all'ovvio problema di ridondanza d'informazioni,

esiste anche il non meno grave problema di allineamento delle due sorgenti in caso di aggiornamento.

La seconda problematica affrontata è stata quella di integrare i due database in un'unica base di dati allo scopo di avere un'unica sorgente di informazioni cui accedere sia per la realizzazione delle pagine Web dinamiche, sia per la stesura della Guida dello studente.

1.3 Schema relazionale

Nel rielaborare lo schema relazionale si è posta particolare attenzione ad una prima e importante suddivisione delle informazioni contenute nei due database: quelle dipendenti e quelle indipendenti dall'anno accademico.

Riportiamo qui di seguito una parte dello schema relazionale finale, comprendente alcuni esempi significativi delle relazioni inrodotte, specificando per ognuna di esse una breve descrizione dei vari attributi.

1.3.1 Relazioni indipendenti dall'anno accademico

La relazione forse più ovvia che riporta informazioni non dipendenti dall'anno accademico è quella che contiene i dati delle varie facoltà dell'universita' degli studi di Modena e Reggio Emilia:

TBL_FACOLTA (IdFacolta , Denominazione , Codseg , Engname , Label , Indirizzo , Sede)
FK : Sede *REFERENCES* TBL_SEDI

Questa relazione contiene un riferimento ad un'altra relazione: TBL_SEDI contenete le varie sedi (Modena o Reggio Emilia) con le relative informazioni anch'esse indipendenti dall'anno accademico.

Un'altra delle relazioni più significative è quella che riporta i vari corsi presenti in una certa facoltà. Essa dipende da altre tre relazioni, quella della facoltà

precedentemente riportata, e quelle delle classi di laurea e delle categorie, che vengono riportate anch'esse qui di seguito.

TBL_CORSI_CLASSI (Numero , Livello , Denominazione)

TBL_CORSI_CATEGORIE (IdCategoria , Denominazione , NuovoOrdinamento , Engname , Sigla)

TBL_CORSI (IdCorso , Denominazione , IdFacolta , IdCategoria , NClasse , LClasse , Codsegr , AnnoDiIstituzione , Label , Engname)
FK : IdFacolta *REFERENCES* FACOLTA
FK : IdCategoria *REFERENCES* TBL_CORSI_CATEGORIE
FK : NClasse , LClasse *REFERENCES* TBL_CORSI_CLASSI

La relazione TBL_CORSI_CLASSI contiene le classi di laurea, sia di I livello che di II livello stabilite dal MURST. La base di dati realizzata contiene tutte le 42 classi di laurea di I livello e le 104 classi di laurea di II livello (classi delle lauree specialistiche). La relazione TBL_CORSI_CATEGORIE riporta le categorie (corso di laurea, corso di laurea specialistico, diploma universitario, ecc.) dei corsi di studio specificando, attraverso l'attributo flag NuovoOrdinamento, se si riferisce al nuovo o al vecchio ordinamento didattico (i corsi di laurea possono essere sia del nuovo che del vecchio ordinamento).

Un esempio di violazione della 3NF sarebbe stata l'introduzione dell'attributo NuovoOrdinamento direttamente in TBL_CORSI poiché si sarebbe introdotta la dipendenza funzionale IdCategoria → NuovoOrdinamento , dovuta al fatto che l'appartenenza al VOD o al NOD dipende unicamente dalla classe di appartenenza del corso.

La fonte da cui si è attinto per ottenere le informazioni da inserire negli opportuni attributi di questa relazione è il RdCL che, per ciascun corso, definisce i seguenti punti:

1) gli obiettivi formativi specifici del corso;

2) il numero minimo di crediti formativi da acquisire, per ciascuna tipologia di attività formativa, per il conseguimento del titolo del corso cui il regolamento si riferisce;
3) gli eventuali curricula offerti;
4) gli insegnamenti con l'indicazione del numero dei crediti, dell'ambito disciplinare, e dei settori scientifico disciplinari (SSD) di riferimento, degli obiettivi formativi specifici, dei contenuti, delle propedeuticità consigliate, delle modalità di svolgimento della didattica e di accertamento del profilo.

Il RdCL viene redatto in un certo anno accademico di riferimento e può rimanere in vigore per diversi anni fino a che non viene sostituito da uno nuovo. Come informazione non dipendente dall'anno accademico si è deciso dunque di mantenere in una apposita relazione anche un riferimento a ciascun regolamento di ciascun corso:

TBL_CORSO_STUDIO_REGOLAMENTO (IdCorso , IdAnnoRiferimento , Denominazione , urlDocumento , Completo)
FK : IdCorso *REFERENCES* CORSI
FK : IdAnnoRiferimento *REFERENCES* TBL_ANNI_ACCADEMICI

L'attributo IdAnnoRiferimento indica l'anno accademico in cui è stato redatto il regolamento di un certo corso, mentre l'attributo urlDocumento indica un URL al quale si può trovare il testo completo del documento stesso.

Le informazioni relative agli insegnamenti previsti per ciascun corso nell'allegato B del corrispondente RdCL vengono mantenute attraverso altre due relazioni, che referenziano a loro volta altre relazioni di ovvio significato:

TBL_INSEGNAMENTI (IdInsegnamento , Denominazione , Engname , Crediti , IdSsd , Obiettivi , Svolgimento , Accertamento , ParoleChiave , IdAmbito , IdAttivita)
FK : IdSsd *REFERENCES* TBL_CORSI_SSD
FK : IdAmbito *REFERENCES* TBL_INSEGNAMENTI_AMBITI
FK : IdAttivita *REFERENCES* TBL_INSEGNAMENTI_ATTIVITA

TBL_REGOLAMENTO_INSEGNAMENTI (IdCorso , IdAnnoRiferimento , IdInsegnamento , AnnoCorso)
FK : IdCorso , IdAnnoRiferimento *REFERENCES* TBL_CORSO_STUDIO_REGOLAMENTO
FK : IdInsegnamento *REFERENCES* TBL_INSEGNAMENTI

Una relazione di interesse particolare è quella che contiene le informazioni relative ai docenti. Essa sarà un'importante sorgente di dati anche per l'applicazione realizzata che verrà descritta nei prossimi capitoli.

TBL_DOCENTI (Id , Nome , Cognome , Telefono , Email , HomePage , IdRuolo , Ricevimento , Fax , Username , Password , cf , FotoType , FotoPath , FotoFile , Pubblicata , PathSplibero , Ricerca , Cooperazioni , Incarichi , Affiliazioni , Collaborazioni , Ufficio)
AK : Username
FK : IdRuolo *REFERENCES* TBL_RUOLO

Un commento particolare meritano gli attributi della relazione TBL_DOCENTI che giocano un ruolo fondamentale nell'applicazione realizzata della homepage dei docenti. Gli attributi Username e Password saranno necessari per l'autenticazione del docente quando si collega alla sua homepage per modificarla. Gli attributi FotoType, FotoPath, FotoFile contengono rispettivamente il *MIME content type* [1] del file della foto del docente, il percorso nel quale è salvato e il nome del file. Sono ammessi valori nulli per il nome e il content type del file, nel caso in cui il docente non volesse visualizzare alcuna foto sulla propria homepage, ma non per il percorso che comunque deve essere specificato dall'amministratore di sistema e messo a disposizione nel caso il docente voglia visualizzare la foto in un secondo momento. Infine l'attributo PathSplibero contiene il percorso specificato dall'amministratore di

[1] Il MIME content type è una sigla standard che viene utilizzata nelle comunicazioni via internet per specificare il tipo di contenuto di un file che viene inviato. Vedi appendice B

sistema, nel quale verranno salvati tutti i files che il docente decide di pubblicare on-line nella parte di spazio libero della sua homepage.

Per completezza si riporta di seguito anche una delle relazioni che erano presenti nello schema relazionle del database della Guida dello studente e che sono state integrate nello schema relazionale finale:

TBL_TITOLETTI (Num , Ordine , Titoletto , Caratteri)

Questa relazione contiene i titoli dei vari paragrafi che si trovano nella presentazione di ciascun corso di ogni facoltà presente nella guida dello studente. Il contenuto di ciascun paragrafo varia di corso in corso e di anno in anno ed è riportato in un'altra relazione (TBL_CORSITITOLETTI).

1.3.2 Relazioni dipendenti dall'anno accademico

La principale fonte di informazioni che varia con l'anno accademico è il Manifesto degli Studi di un certo corso. Esso viene redatto anno per anno e deve far riferimento al RdCL attualmente in vigore.

Ogni corso che viene attivato in un certo anno accademico deve avere un Manifesto degli Studi associato che dunque si è deciso di riportare in una apposita relazione assieme ad altre informazioni interne relative al corso stesso e variabili di anno in anno, come ad esempio il Presidente e il Referente:

TBL_MANIFESTO (IdCorso , IdAnno , IdAnnoRegolamento , IdPresidente , IdReferente , IdCordinatore , Aregime , Completo , Durata , IdAccesso , Posti , Homepage , NumTel , NumFax , Email , Notalibera , Ultimoagg)
FK : IdCorso , IdAnnoRegolamento
REFERENCES TBL_CORSO_STUDIO_REGOLAMENTO
FK : IdAnno *REFERENCES* TBL_ANNI_ACCADEMICI
FK : IdPresidente *REFERENCES* TBL_DOCENTI
FK : IdReferente *REFERENCES* TBL_DOCENTI
FK : IdCoordinatore *REFERENCES* TBL_DOCENTI
FK : IdAccesso *REFERENCES* TIPOACCESSO

Si noti che ogni manifesto fa riferimento ad un certo regolamento che deve essere l'attuale RdCL del corso cui si riferisce.

In ciascun manifesto devono poi essere riportati tutti gli insegnamenti effettivamente attivati tra quelli previsti dal relativo RdCL. Questo elenco viene mantenuto in una relazione apposita che assume un ruolo fondamentale nello sviluppo dell'applicazione delle homepage dei docenti:

TBL_INSEGNAMENTI_AA (IdCorso , IdAnnoManifesto , IdInsegnamento , DenPeriodo , Programma , TestiConsigliati , Note , Pubblicata , VODProp , Ascelta , IdOrientamento , PathMatdid , UsernameDefault , PasswordDefault , DefaultFlag , Username , Password)
FK : IdInsegnamento *REFERENCES* TBL_INSEGNAMENTI
FK : IdCorso , IdAnnoManifesto *REFERENCES* TBL_MANIFESTO
FK : IdCorso, IdAnnoManifesto , DenPeriodo *REFERENCES* TBL_CORSI_AA_PERIODI
FK : IdCorso , IdAnnoManifesto , IdOrientamento *REFERENCES* TBL_ORIENTAMENTI_AA

Si rende necessario un breve chiarimento sul contenuto di alcuni attributi che saranno utilizzati nella applicazione e il cui significato verrà approfondito in seguito. L'attributo Pubblicata è un flag che stabilisce se il docente ha deciso di pubblicare o meno la pagina del materiale didattico dell'insegnamento. L'attributo PathMatdid contiene il percorso fornito dall'amministratore di sistema nel quale saranno salvati i files che il docente deciderà di mettere on-line come materiale didattico. Gli attributi UsernameDefault, PasswordDefault, Usename, Password e DefaultFlag saranno utilizzati per autenticare gli studenti che intendono accedere al materiale pubblicato.

Per sapere in un certo anno accademico i docenti di un certo insegnamento si è mantenuta una relazione apposita:

TBL_DOCENTI_INSEGNAMENTI_AA (IdCorso , IdAnnoManifesto , IdInsegnamento , IdDocente , Lettere , Titolare)

FK : IdCorso , IdAnnoManifesto , IdInsegnamento *REFERENCES* TBL_INSEGNAMENTI_AA
FK : IdDocente *REFERENCES* TBL_DOCENTI

E' prevista la possibilità che un insegnamento venga tenuto da più docenti diversi, ad esempio nel caso di suddivisione delle aule per lettere (dalla 'A' alla 'L' in una aula e dalla 'M' alla 'Z' in un'altra).

Infine si riportano le relazioni che sono state introdotte appositamente per la gestione del materiale da pubblicare on-line, sia facente parte dello spazio libero all'interno dell'homepage, sia facente parte del materiale didattico di un certo insegnamento. Si è pensato di organizzare questo materiale in modo che il docente possa specificare una serie di argomenti identificati da un titolo, ciascuno dei quali contenga un elenco di files, un elenco di links e un elenco di note. Essendo particolare oggetto di interesse dell'applicazione si rimanda ad ulteriori dettagli chiarificatori ai capitoli successivi.

TBL_SPLIBERO_TITOLI (IdDocente , IdTitolo , Denominazione , Ordine)
FK : IdDocente *REFERENCES* TBL_DOCENTI

TBL_ SPLIBERO_FILE (IdDocente , IdTitolo , IdFile , NomeFile , FileDescrizione , ContentType , Ordine)
FK : IdDocente , IdTitolo *REFERENCES* TBL_SPLIBERO_TITOLI

TBL_ SPLIBERO_LINK (IdDocente , IdTitolo , IdLink , Url , UrlDescrizione , Ordine)
FK : IdDocente , IdTitolo *REFERENCES* TBL_SPLIBERO_TITOLI

TBL_ SPLIBERO_NOTE (IdDocente , IdTitolo , IdNota , NotaDescrizione , Ordine)
FK : IdDocente , IdTitolo *REFERENCES* TBL_SPLIBERO_TITOLI

TBL_MATDID_TITOLI (IdCorso , IdAnnoManifesto , IdInsegnamento , IdTitolo , Denominazione , Ordine)
FK : IdCorso , IdAnnoManifesto , IdInsegnamento *REFERENCES*

TBL_INSEGNAMENTI_AA

TBL_MATDID_FILE (IdCorso , IdAnnoManifesto , IdInsegnamento , IdTitolo , IdFile , NomeFile , FileDescrizione , ContentType , Ordine)
FK : IdCorso , IdAnnoManifesto , IdInsegnamento , IdTitolo
REFERENCES TBL_MATDID_TITOLI

TBL_MATDID_LINK (IdCorso , IdAnnoManifesto , IdInsegnamento , IdTitolo , IdLink , Url , UrlDescrizione , Ordine)
FK : IdCorso , IdAnnoManifesto , IdInsegnamento , IdTitolo
REFERENCES TBL_MATDID_TITOLI

TBL_MATDID_NOTE (IdCorso , IdAnnoManifesto , IdInsegnamento , IdTitolo , IdNota , NotaDescrizione , Ordine)
FK : IdCorso , IdAnnoManifesto , IdInsegnamento , IdTitolo
REFERENCES TBL_MATDID_TITOLI

1.4 Implementazione del database

Il database finale ottenuto integrando i due database esistenti normalizzati e aggiungendo le nuove tabelle necessarie all'applicazione sviluppata è stato implementato su SQL Server.

Si riportano a titolo esemplificativo gli statement di creazione delle tabelle relative al materiale didattico dei vari insegnamenti nei diversi anni accademici:

```
create table TBL_MATDID_TITOLI(
IdTitolo int,
IdCorso int,
IdInsegnamento int,
IdAnnoManifesto int,
Denominazione varchar(100) not null,
Ordine int,
primary key
(IdTitolo,IdCorso,IdInsegnamento,IdAnnoManifesto),
foreign key (IdCorso,IdInsegnamento,IdAnnoManifesto)
```

```
references TBL_INSEGNAMENTI_AA
)

create table TBL_MATDID_FILE(
IdFile int,
IdTitolo int,
IdCorso int not null,
IdInsegnamento int not null,
IdAnnoManifesto int not null,
NomeFile varchar(100) not null,
FileDescrizione varchar(100) not null,
ContentType varchar(50) not null,
Ordine int,
primary key (IdFile,IdTitolo),
foreign key
(IdTitolo,IdCorso,IdInsegnamento,IdAnnoManifesto) references
TBL_MATDID_TITOLI
)

create table TBL_MATDID_LINK(
IdLink int,
IdTitolo int,
IdCorso int not null,
IdInsegnamento int not null,
IdAnnoManifesto int not null,
Url varchar(100) not null,
UrlDescrizione varchar(100) not null,
Ordine int,
primary key (IdLink,IdTitolo),
foreign key
(IdTitolo,IdCorso,IdInsegnamento,IdAnnoManifesto) references
TBL_MATDID_TITOLI
)

create table TBL_MATDID_NOTE(
IdNota int,
IdTitolo int,
IdCorso int not null,
IdInsegnamento int not null,
IdAnnoManifesto int not null,
NotaDescrizione varchar(200) not null,
Ordine int,
```

```
primary key (IdNota,IdTitolo),
foreign key
(IdTitolo,IdCorso,IdInsegnamento,IdAnnoManifesto) references
TBL_MATDID_TITOLI
)
```

1.5 Inconsistenze

Uno schema relazionale ben progettato, oltre a non violare la 3NF, dovrebbe riuscire ad esprimere la maggior parte dei vincoli imposti dalle specifiche dei requisiti, attraverso l'introduzione dei cosiddetti vincoli di integrità. Ad esempio il vincolo che impone che ogni docente abbia un unico username è stato imposto dichiarando il relativo attributo come chiave alternativa. Un altro esempio è il vincolo che impone che un certo Manifesto sia attivato con riferimento ad uno e uno solo RdCL tra quelli inseriti, realizzato imponendo (IdCorso , IdAnnoManifesto) come chiave primaria e (IdCorso , IdAnnoRegolamento) come foreign key referenziante la relazione TBL_CORSO_STUDIO_REGOLAMENTO.

Esistono però dei vincoli particolari che non possono essere espressi con soli vincoli di integrità. E' questo il caso della relazione TBL_INSEGNAMENTI_AA nella quale possono essere inserite tuple inconsistenti, infatti nulla impedisce l'inserimento di un insegnamento facente parte di un RdCL diverso da quello associato al Manifesto cui l'insegnamento si riferisce.

Per ovviare a queste inconsistenze la maggior parte dei DBMS relazionali mette a disposizione strumenti, quali i *triggers*, che sono particolari procedure batch che vengono attivate automaticamente dal sistema su particolari eventi, e che consentono di effettuare controlli sugli aggiornamenti dei dati del database.

SQL Server fornisce supporto ai trigger e anche alle cosiddette *Stored Procedures*, cioè insiemi di statement SQL preventivamente memorizzati in una procedura batch successivamente compilata. Queste ultime possono essere utili per

gestire operazioni ripetitive che possono essere eseguite semplicemente richiamandole tramite nome ed eventuali parametri. Un esempio di Stored Procedure potrebbe essere quella per cancellare un intero argomento con tutto il suo contenuto dal materiale didattico di un insegnamento: se si tentasse di cancellare una tupla da TBL_MATDID_TITOLI senza aver prima cancellato tutte le tuple dalle altre tabelle che la riferiscono, SQL Server darebbe un errore. Si può dunque realizzare una procedura che cancelli prima tutte le eventuali tuple referenti, e poi la tupla riferita.

Questi strumenti sono utili agli sviluppatori per assicurarsi che chi ha accesso al DBMS non possa creare inconsistenze. In genere però la maggior parte degli utenti accede ai dati attraverso degli opportuni front-end che sono in grado di effettuare tutti i controlli del caso. Inoltre si tratta di tecnologie fortemente dipendenti dal DBMS utilizzato. Per questi motivi si è deciso di non implementare né triggers né stored procedures, ma di assegnare questi compiti al front-end dell'applicazione.

Per completezza si riporta comunque qui di seguito il trigger progettato, ma non implementato, per eliminare le inconsistenze sugli inserimenti nella tabella TBL_INSEGNAMENTI_AA.

```
/* TRIGGER TRINSEGNAMENTO */

create trigger TRINSEGNAMENTO
on TBL_INSEGNAMENTI_AA
for insert

as

select @IdCS=IdCorso , @IdAM=IdAnnoManifesto ,
@IdI=IdInsegnamento , @IdAR=IdAnnoRegolamento
from INSERTED i , TBL_MANIFESTO c
where i.IdCorso=c.IdCorso  and  i.IdAnnoManifesto=c.IdAnno

if not exists ( select * from
TBL_REGOLAMENTO_INSEGNAMENTI
                        where  IdCorso=@IdCS
                         and   IdAnnoRiferimento=@IdAR
```

```
                        and    IdInsegnamento=@Idl
                )
begin

   rollback transaction

end

go
```

Capitolo 2

La piattaforma J2EE

Il World Wide Web e Internet rappresentano le fondamenta sulle quali le imprese tendono a lavorare sempre più intensamente per costruire un nuovo tipo di economia: l'*economia dell'informazione (information economy).* In questa economia le informazioni acquistano un valore talmente alto da essere paragonabile a quello dei classici beni e servizi, e diventano parte vitale del mercato. Esse forniscono un valore strategico di grande portata per un'impresa, tanto da poter costituire un punto chiave per ottenere competitività.

L'involucro nel quale le imprese trasmettono queste informazioni come una qualsiasi altra merce, è costituito dall'insieme di quelle che vengono chiamate *applicazioni su misura distribuite (distribuited custom applications).* E' compito degli sviluppatori di queste applicazioni capire quali sono le specifiche esigenze dei possibili utenti di queste informazioni e fornire dunque funzionalità adeguate. E' altresì fondamentale nello scenario altamente competitivo dell'information economy avere un buon response time, cioè progettare applicazioni in grado di potersi adattare alle esigenze delle imprese nel modo più veloce possibile man mano che si rendessero necessari interventi di aggiornamento.

L'obiettivo della piattaforma *Java 2 Enterprise Edition (J2EE)* [J2EE] , distribuita dalla Sun Microsystem[SUN] , è quello di definire delle funzionalità standard che aiutino a sviluppare queste applicazioni distribuite, utilizzando un vasto range di nuove tecnologie e un modello di progettazione component-based.

2.1 Vantaggi della piattaforma J2EE

La piattaforma J2EE offre una serie di benefici agli sviluppatori di applicazioni distribuite.

2.1.1 Architettura e sviluppo semplificati

La J2EE supporta un modello di sviluppo *component-based* semplificato. Essendo basata sul linguaggio di programmazione Java e sulla Java 2 Standard Edition (J2SE platform), questo modello offre grandi vantaggi di portabilità (sistema *Write Once, Run Anyway*) e garantisce il supporto su tutti i server conformi a questo standard.

Il modello di sviluppo component-based J2EE arricchisce la produttività delle applicazioni in diversi modi:

1) assicura flessibilità alle funzionalità desiderate, consentendo diverse possibilità di configurazione dell'architettura della applicazione, a seconda di vari fattori, come ad esempio il tipo di client, la sicurezza richiesta per gli accessi alle sorgenti di dati, e altri che verranno approfonditi in seguito. Il modello component-based inoltre semplifica la manutenzione dell'applicazione, in quanto i singoli componenti possono essere aggiornati o sostituiti indipendentemente.
2) assicura ai componenti la disponibilità di una serie di servizi nell'ambiente di run-time, e la possibilità di essere dinamicamente connessi ad altri componenti, semplicemente fornendo opportune interfacce. Ad esempio attraverso i *deployment descriptors*[2] è possibile comunicare specifici parametri all'ambiente di run-time personalizzando l'applicazione senza dover modificare il codice dei componenti.
3) supporta la suddivisione dei compiti, in quanto ciascun set di componenti può essere associato ad un certo ambito di sviluppo, consentendo a ciascuna figura

[2] Il deployment descriptor è un file di testo in formato XML che specifica il comportamento di un componente attraverso tag XML. Vedi paragrafo 2.6.

professionale di concentrarsi specificatamente sulle proprie competenze e abilità. Questa suddivisione dei compiti, oltre a favorire la specializzazione e quindi migliorare la qualità dei singoli componenti, consente un criterio di sviluppo dell'applicazione in parallelo, aumentandone la velocità di produzione e manutenzione.

2.1.2 Scalabilità

La J2EE fornisce supporti per adattare in modo semplice e funzionale una applicazione ad eventuali aumenti o riduzioni delle dimensioni del progetto senza comprometterne l'efficienza delle prestazioni. Ad esempio il supporto per il *connection pooling*[3] assicura rapido accesso ai dati anche ad un numero crecente di clients.

Inoltre l'architettura distribuita che la J2EE supporta consente di ottenere un vantaggioso sistema di bilanciamento del carico di lavoro, permettendo la configurazione dei vari livelli dell'architettura per la gestione su server diversi.

2.1.3 Integrazione su sorgenti di informazione già esistenti

La J2EE platform, assieme alla J2SE platform, include una serie di *API (Application Program Interface)* standard per accedere alle varie sorgenti di informazione esistenti nell'impresa.

Si riporta l'elenco di queste API.

- *JDBC (Java DataBase Connectivity)* è l'API per la connessione ai database relazionali. Essa fornisce un interfaccia a livello di applicazione usata nel codice dei componenti per accedere al database in modo indipendente dal particolare DBMS utilizzato; sarà poi necessario utilizzare un driver specifico che si ponga come interfaccia tra l'API JDBC e lo specifico DBMS utilizzato. Grazie a questo meccanismo a due livelli è possibile mantenere il codice dei componenti

dell'applicazione indipendente dal DBMS che può essere sostituito con un altro semplicemente cambiando il driver, e senza necessità di riscrittura del codice.

- *JTA (Java Transaction API)* è l'API per la gestione e il coordinamento delle transazioni attraverso sorgenti di informazioni transazionali eterogenee.
- *JNDI (Java Naming and Directory Interface)* è l'API per accedere ad informazioni attraverso un sistema di nomi e percorsi, utilizzando così lo stesso meccanismo utilizzato per riferire i files nel File System.
- *JMS (Java Message Service)* è l'API per inviare e ricevere messaggi attraverso sistemi di enterprise-messaging come l'IBM MQ Series e TIBCO Rendez-vous che richiedono l'attivazione del servizio attraverso un JMS provider.
- *JavaMail* è l'API utilizzata dai componenti per inviare e ricevere email.
- *Java IDL* è l'API per richiamare oggetti CORBA remoti, attraverso il protocollo IIOP. Questi oggetti sono tipicamente esterni alla J2EE e possono essere scritti in qualunque linguaggio.

Oltre a questi servizi che già sono presenti nell'attuale versione della J2EE platform, ve ne sono altri in via di sviluppo che saranno aggiunti nelle successive versioni attraverso l'architettura dei *Connectors*.

2.1.4 Scelta di servizi, tools e componenti

Il marchio J2EE è punto centrale per creare un mercato di servizi, tools e componenti tutti conformi allo stesso standard.

Le organizzazioni che sviluppano applicazioni J2EE possono contare su una varietà sempre crescente di fornitori di piattaforme J2EE con diverse possibilità di scelta sia a livello hardware, sia a livello di sistemi operativi e configurazioni di server, a seconda degli obiettivi e strategie adottate.

I vari componenti J2EE, con particolare riferimento agli EJB e alle JSP di cui si tratterà dettagliatamente in seguito, sono stati ideati per essere facilmente manipolati

[3] Il pool di connessioni è un meccanismo molto utile per diminuire il gravoso tempo di accesso ai

da tool grafici che consentono di automatizzare tanti dei tradizionali compiti richiesti, come ad esempio il debugging. Gli sviluppatori J2EE hanno così la possibilità di scegliere il tool conforme allo standard J2EE che meglio si addice alle loro esigenze.

Grazie al modello component-based si ha poi la possibilità di sviluppare componenti conformi allo standard che possono essere così riutilizzati da una qualsiasi applicazione J2EE.

2.1.5 Modello di sicurezza semplificato e unificato

Gli sviluppatori possono specificare i requisiti di sicurezza di un componente. Attraverso un sistema dichiarativo, sia gli EJB-component sia i Web-component possono specificare attraverso i deployment descriptor i criteri per associare i vari livelli di accesso a diversi ruoli e quindi alle diverse categorie di utenti. Per gli EJB esiste addirittura la possibilità di definire ruoli diversi per ciascun metodo.

Le specifiche J2EE incoraggiano l'utilizzo di questo sistema dichiarativo per consentire al codice dei componenti di rimanere totalmente indipendente dai vincoli legati alla sicurezza. Ciò non sempre è possibile, poiché in taluni casi è necessario creare livelli e vincoli di sicurezza che non possono essere espressi con un semplice mapping tra ruoli e utenti. Perciò viene comunque mantenuta la possibilità di inserire security-checks anche all'interno del codice attraverso opportune API.

2.2 Architettura multilivello della J2EE

Si analizza ora nel dettaglio il modello di applicazione distribuita multilivello della J2EE platform.

Si è già accennato alla caratteristica component-based del modello J2EE con i vantaggi che ne derivano. La logica di ogni applicazione J2EE può essere infatti suddivisa in più componenti in accordo con la funzione da essi svolta, ciascuno dei

database. Vedi paragrafo 2.7

quali può essere installato in una macchina differente a seconda del livello logico di appartenenza all'interno dell'architettura J2EE.

La architettura multilivello J2EE individua tre livelli fondamentali: il *client-tier*, il *middle-tier* e l'*EIS-tier.*

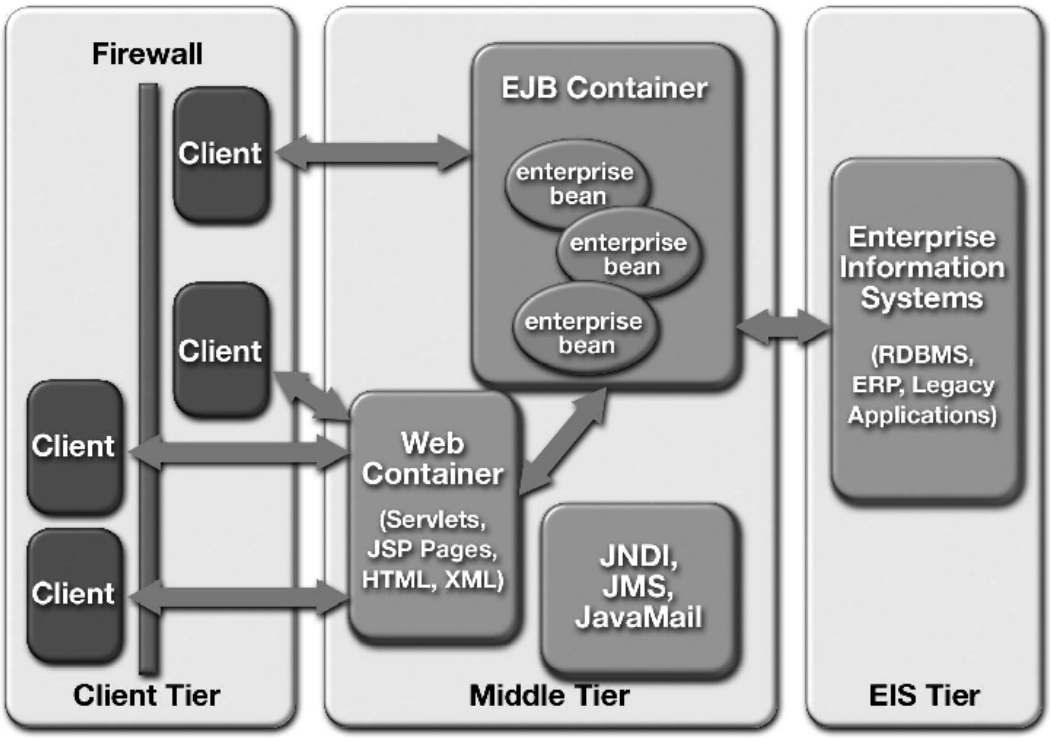


fig. 2.1 - Architettura J2EE -

La figura mostra i vari componenti e sevizi che costituiscono un tipico ambiente J2EE. La presenza di un livello intermedio tra il client e le sorgenti di informazione fornisce alla J2EE platform i tipici vantaggi derivanti da un'architettura multilivello.

Le tipiche architetture client-server a due livelli, nelle quali il server è in pratica costituito dalle sorgenti di informazione, hanno dato prova di essere fortemente limitanti a causa della necessità di installare e mantenere un front-end completo di interfaccia grafica e di logica applicativa direttamente su ciascun client. Ciò

naturalmente si traduce in un forte carico di lavoro sulla macchina client, creando problemi derivati dalla limitatezza delle risorse di tali macchine, tipicamente dei semplici PC, e nel non meno grave problema della manutenzione dell'applicazione: ogni più piccolo cambiamento nel codice dell'applicazione deve essere ripetuto su tutti i client.

Con un'architettura a tre livelli questi problemi vengono superati grazie all'introduzione di un middle tier che si pone tra client e la sorgente di informazioni. In questo scenario il server può ospitare sia il middle tier che l'EIS tier, oppure questi ultimi possono essere ospitati su server differenti, dando origine al middle tier server e all'EIS server.

Il middle tier server si occupa di gestire le richieste dei vari client, di reperire le opportune informazioni dall'EIS server, rielaborarle secondo la logica applicativa, e fornirle al client, che si trova così notevolmente alleggerito.

La J2EE fornisce uno standard ben preciso per implementare il middle tier server, e fornisce indicazioni e raccomandazioni per implementare gli altri due livelli.

Si rimanda ai prossimi paragrafi la descrizione dettagliata dei tre livelli della J2EE, mentre in questa sede si vuole introdurre un altro elemento chiave dell'architettura: i *containers*.

I containers sono degli ambienti run-time standardizzati in grado di fornire specifici servizi ai vari componenti. Ciascun componente può contare sul supporto del container in cui è inserito per ottenere gli stessi servizi standard in tutte le piattaforme J2EE qualunque sia lo specifico provider. I vantaggi che ne derivano sono i seguenti:

- gli sviluppatori possono disinteressarsi di come i container gestiscono tali servizi, potendo così concentrarsi sulla funzionalità del codice.
- attraverso i deployment descriptor, che fungono da interfaccia tra i componenti e il container si possono specificare i parametri e i comportamenti dei componenti stessi al momento del deployment dell'applicazione senza dover interferire sul codice.

2.3 EIS (Enterprise Information Systems) tier

E' il livello più basso dell'architettura, cioè la parte di dati memorizzati in modo permanente con varie tipologie di risorse, tipicamente RDBMS. Questi ultimi possono supportare accessi controllati attraverso le transazioni. Essi possono partecipare in transazioni assieme ad altre risorse transazionali in un sistema di database distribuito attraverso il protocollo *two-phase commit*, gestito da un transaction manager supportato dal J2EE server.

Questa integrazione di sorgenti di informazione funziona bene quando queste sono omogenee. Purtroppo non sempre è così quando si sviluppa una applicazione distribuita che si deve appoggiare a varie sorgenti.

E' attualmente in via di sviluppo da parte dei progettisti della Sun un nuovo tipo di API, le *API J2EE Connector*, allo scopo di definire uno standard per connettere la J2EE platform con sorgenti di informazioni eterogenee.

2.4 Middle tier

I maggiori benefici del modello di applicazioni J2EE si riscontrano nel middle tier. Esso è ulteriormente scomposto in due sottolivelli: un *Web-tier* e un *EJB-tier* che possono trovarsi anche su differenti hosts, oppure sullo stesso host ma in differenti Java Virtual Machines, o infine anche sulla stessa JVM.

Le funzioni di business logic della applicazione vengono implementate attraverso componenti *Enterprise Java Beans (EJB)* all'interno dell'EJB-tier.

Al Web-tier spetta invece il compito di "presentare" al client (in questo caso un Web-client) i risultati dell'elaborazione della business logic, attraverso la generazione di pagine web dinamiche.

2.4.1 EJB tier

La tecnologia degli *Enterprise Java Beans (EJB)* [SPECEJB] offre un modello component-based distribuito che consente agli sviluppatori di concentrare la loro abilità sui business problem dell'applicazione, lasciando al *EJB container* il compito di gestire tutta una serie di servizi di sistema a basso livello. Questa separazione di ruoli, punto di forza dell'architettura J2EE che si ritrova anche nel Web tier, permette un più rapido sviluppo di applicazioni altamente scalabili, flessibili e sicure.

Gli EJB sono componenti che costituiscono un collegamento fondamentale tra i componenti del Web tier ai quali viene affidato esclusivamente il compito di gestire la presentation logic, e le risorse di informazione mantenute nel EIS tier.

La Business logic e i business-objects

E' difficile dare una definizione rigorosa di business logic di una applicazione. In senso lato può essere definita come un insieme di direttive per gestire ogni specifica funzione che l'applicazione deve svolgere.

Volendo adottare un approccio object-oriented, una funzione può essere scomposta in un insieme di componenti, o elementi chiamati *business-objects*. Come gli altri elementi, anche questi sono dotati di una specifica struttura dati e di un determinato comportamento. Ad esempio un impiegato può essere modellato con un oggetto-impiegato che ha una struttura dati, rappresentata da un insieme di attributi quali il nome, l'indirizzo, il codice fiscale e così via; inoltre ha dei metodi ad es. per assegnarlo ad un nuovo dipartimento, aumentargli lo stipendio, ecc.

In modo più rigoroso si può allora definire la business-logic come l'insieme delle regole che servono ad identificare struttura e comportamento dei business-objects, e i criteri di interazione con gli altri oggetti dell'applicazione.

Requisiti comuni dei business-objects

Si riportano qui di seguito alcuni requisiti comuni ai business object.

1) *Mantenere lo stato*. I business-objects solitamente necessitano di mantenere, in modo conversazionale o permanente, lo stato rappresentato dalle variabili di istanza tra le invocazioni dei vari metodi.
2) *Operare su dati condivisi*. La condivisione di certe risorse deve essere gestita attraverso il controllo della concorrenza e un appropriato livello di isolation dei dati condivisi, in modo da assicurarne la consistenza.
3) *Partecipare alle transazioni*. L'atomicità di una transazione deve essere garantita anche se questa si distribuisce su diverse risorse remote.
4) *Servire un gran numero di client*. Un business-object deve essere disponibile in istanze multiple a diversi client contemporaneamente. Il meccanismo di gestione di queste istanze deve essere in grado di fornire a ciascun client un business-object disponibile a servire la sua richiesta. Senza un tale meccanismo il sistema potrebbe eventualmente esaurire le risorse e non essere più in grado di servire ulteriori richieste.
5) *Fornire accesso remoto ai dati*. I client devono poter accedere alle risorse dei business-objects anche da remoto attraverso la rete.
6) *Controllare gli accessi*. I servizi offerti dai business-objects spesso richiedono l'autenticazione del client che ne fa richiesta, per consentire l'accesso a risorse protette ai soli client autorizzati.
7) *Garantire la riusabilità*. E' questo un requisito comune a tutti i tipi di oggetti in un approccio object-oriented.

Gli Enterprise Java Beans e l'EJB Container

Nell'architettura J2EE i business-objects vengono modellati con componenti Enterprise Java Beans. Come già accennato essi possono contare sul supporto fornito dall'EJB container che ne gestisce il ciclo di vita e fornisce loro una varietà di servizi. Quando un client invoca un'operazione su un EJB questa viene prima intercettata dal container che può in tal modo gestire i vari servizi che devono eventualmente propagarsi tra diverse sottochiamate ad altri componenti dell'EJB tier.

Grazie all'intercessione del container è anche possibile definire determinati comportamenti del componente al momento del deployment a seconda delle esigenze senza dover effettuare alcun cambiamento nel codice e ricompilazione, ma semplicemente configurandoli attraverso il deployment descriptor.

Il Bean Provider, cioè la figura professionale che si occupa di sviluppare questi componenti, deve fornire anche una "client view" del componente stesso attraverso la definizione di due interfacce: la *home interface* e la *remote interface*. Questo assicura che il client abbia una visione semplificata del componente e che possa disinteressarsi dei dettagli implementativi. Sarà compito del container implementare queste interfacce, crearne le istanze e gestirne il ciclo di vita.

La *home interface* fornisce i metodi per creare e rimuovere gli EJB. Essa deve estendere l'interfaccia `javax.ejb.EJBHome` . Attraverso la home interface il client può inoltre ottenere informazioni sui meta-dati dell'EJB . Per particolari tipi di EJB, di cui si parlerà in seguito, la home interface consente anche di ottenre un handle, cioè un riferimento, alla interfaccia stessa che può essere serializzato e salvato in modo permanente, e infine consente di recuperare istanze particolari del bean salvate in precedenza.

La *remote interface* definisce la vera e propria "client view" dell'EJB, cioè il set di business methods disponibili ai clients. Essa deve estendere l'interfaccia `javax.EJB.EJBObject` .

Per completare lo sviluppo di un EJB occorre fornire anche l'implementazione attuale dei business methods del bean. Ciò viene fatto fornendo la *Enterprise Bean class* che deve contenere il codice di tutti i metodi dichiarati nella remote interface e inoltre deve contenere un metodo `ejbCreate` per ciascun metodo create della home interface. Quando il client invoca un metodo della remote interface, il container lo intercetta e richiama il corrispondente metodo della classe del bean.

L'Enterprise Bean class deve estendere la interfaccia *javax.ejb.EntityBean* oppure *javax.ejb.SessionBean* , a seconda del tipo di bean che implementano.

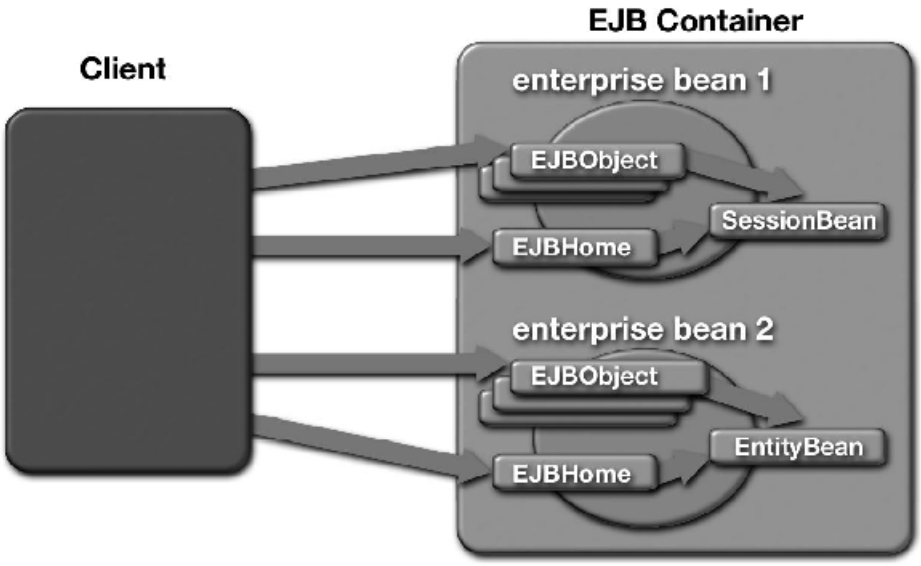


fig 2.2 - Implementazione della client view di un EJB -

Session Beans

Come suggerisce il nome, un *session bean* è simile ad una sessione di interazione con un particolare client. Infatti un session bean rappresenta un unico client all'interno del J2EE server, e non è condiviso da nessun altro client. Esso non è persistente, cioè quando termina la sessione del client, ogni suo riferimento è perso e le risorse associate ad esso sono rilasciate.

Esistono due categorie di session bean: lo *stateful session bean* e lo *stateless session bean*.

Lo *stateful session bean* contiene lo stato conversazionale del client, che significa che lo stato viene mantenuto per tutta la durata della sessione. Per stato conversazionale si intende non solo i valori delle variabili di istanza, ma anche gli oggetti raggiungibili attraverso tali variabili.

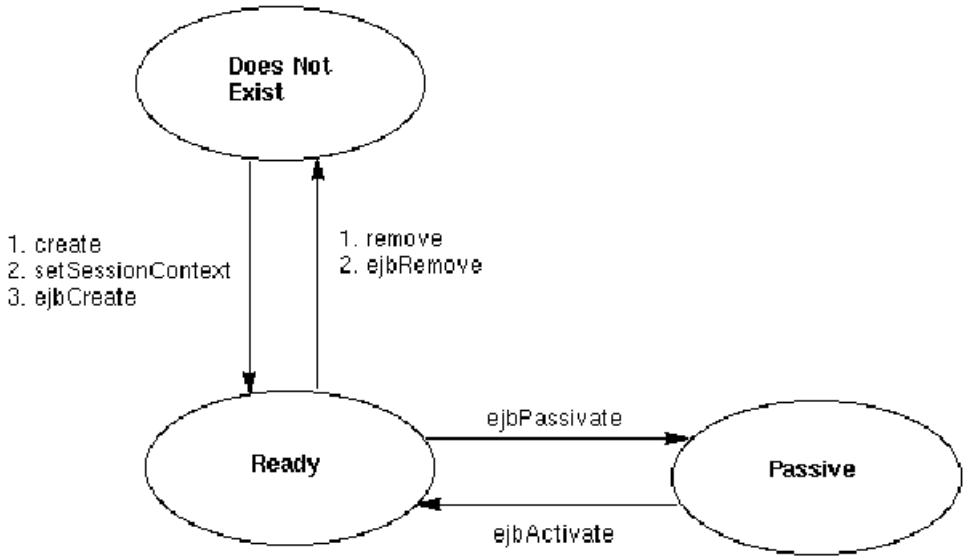


fig 2.3 - Ciclo di vita di un stateful session bean -

La figura 2.3 mostra il ciclo di vita dello stateful session bean. Il client inizia il ciclo di vita richiamando il metodo create della home interface. Il contanier intercetta questa invocazione e provvede a effettuare altre due operazioni: invoca il metdod `setSessionContext` che setta i parametri della sessione nel bean (ad esempio gli eventuali parametri di autenticazione) e richiama il metodo `ejbCreate` della classe del bean corrispondente a quello invocato dal client. A questo punto il bean si trova nello stato di "pronto" a ricevere le invocazioni dei suoi metodi. Quando il client invoca un metodo della remote interface, questo viene intercettato dal container che invoca il corrispondente metodo della classe del bean e il bean rimane in stato "pronto".

Per non appesantire troppo il sistema il container può decidere di liberare la memoria dagli stateful session bean che si trovano in stato di pronto ma non vengono riferiti da un certo periodo di tempo, e salvarli in memoria secondaria, passandoli così allo stato "passivo" attraverso il metodo `ejbPassivate`. Tipicamente il container utilizza un algoritmo least-recently-used per decidere quale bean passare in

stato “passivo”. Se viene richiamato un metodo di un bean che si trova allo stato “passivo”, allora il container provvederà prima a riattivarlo richiamando il metodo `ejbActivate` facendolo così ritornare allo stato “pronto”, e poi a richiamare il metodo.

Prima di terminare la sessione il client invoca il metodo `remove` della home interface e il container, dopo averlo intercettato, provvede a deallocare il bean dalla memoria richiamando il metodo `ejbRemove`.

Lo *stateless session bean* non mantiene lo stato conversazionale del client. Quando un client invoca un metodo di un stateless session bean, le sue variabili di istanza possono contenere uno stato, ma solo per la durata dell’invocazione. Tranne che durante l’invocazione di un metodo tutte le istanze del bean sono equivalenti, lasciando al container il compito di assegnare un’istanza ad ogni client. La home interface di questo tipo di bean può contenere solo un metodo create senza argomenti.

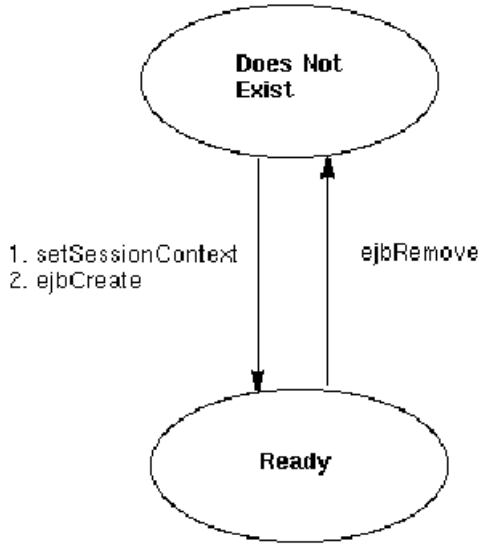


fig 2.4 - Ciclo di vita di un stateless session bean -

La figura 2.4 mostra il ciclo di vita di uno stateless session bean. Come si nota non esiste lo stato "passivo" in quanto non vi è necessità di salvare questo tipo di bean in memoria persistente, perché le risorse occupate devono essere mantenute per un periodo di tempo molto più breve rispetto al stateful session bean.

Inoltre il client non ha accesso ad un metodo `remove` poiché la rimozione del bean avviene solo da parte del container.

Entity Beans

Un entity bean differisce da un session bean per i seguenti aspetti:

- è persistente, nel senso che viene mantenuto lo stato su un database anche al di fuori del ciclo di vita della applicazione e dei processi del J2EE middle server. Ci sono due possibili tipi di persistenza che possono essere dichiarati nel deployment descriptor: *bean-menaged-persistence (BMP)* e *container-menaged-persistence (CMP)*. La prima impone allo sviluppatore di prevedere nel codice tutte le chiamate al database con opportuni statement SQL necessari al container per gestire il ciclo vita del bean; ad esempio il metodo ejbCreate richiamato dal container invierà lo statement SQL di insert opportunamente fornito dallo sviluppatore. La seconda consente invece allo sviluppatore di non curarsi delle chiamate al database, delle quali si occupa automaticamente il container senza necessità di includere alcun statement SQL nel codice.
- consente accessi concorrenti da parte di più client. A tal scopo è importante che si utilizzino entity beans sempre all'interno di transazioni, che possono essere gestite automaticamente dal sistema.
- ha un identificatore unico (primary key).

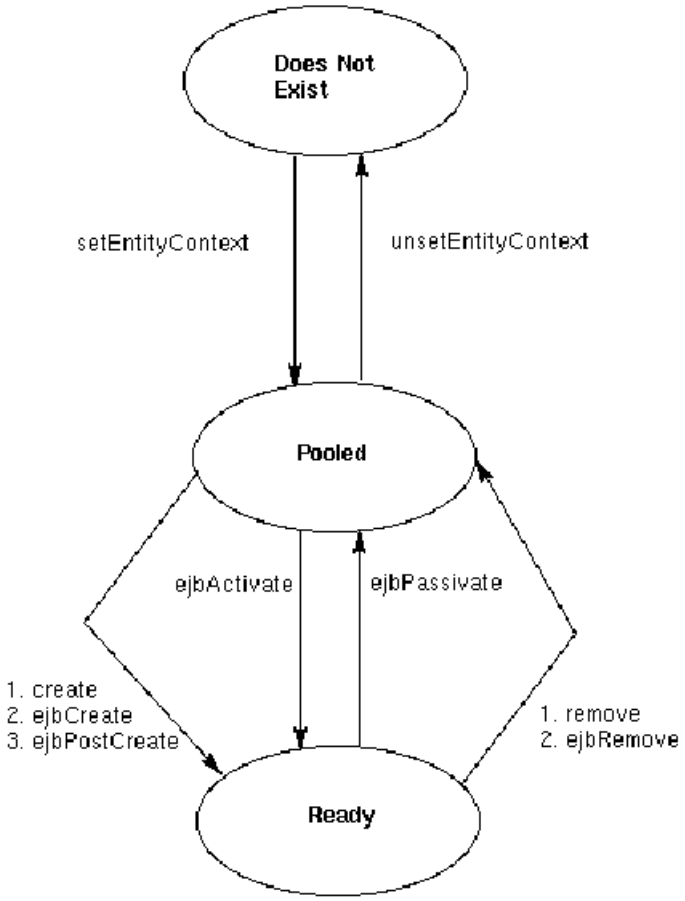


fig 2.5 - Ciclo di vita di un entity bean -

La figura 2.5 riporta il ciclo di vita di un entity bean. Nel momento della partenza del server, tutti gli entity bean dei quali è stato fatto il deployment vengono caricati in memoria in un pool di entity bean, che si trovano dunque in uno stato "pooled". In questo stato tutte le variabili assumono il loro valore di default ed il container assegna l'entity context col metodo `setEntityContext`. A questo punto ci sono due modi possibili in cui il bean può passare dallo stato "pooled" a quello "pronto". Il primo si ha quando il bean viene creato dal client con l'invocazione del metodo create: in questo caso in realtà non viene creato, ma viene prelevato un bean disponibile dal pool ed assegnato al client; successivamente il container richiama il metodo `ejbCreate` corrispondente e il metodo

`ejbPostCreate` per terminare la fase di inizializzazione. Il secondo modo si ha quando il container invoca il metodo `ejbActivate` per recuperare un bean che era stato deallocato dalla memoria primaria. Esistono anche due percorsi che portano dallo stato "pronto" allo stato "pooled" e sono per invocazione del metodo `ejbPassivate` da parte del container o del metodo `remove` da parte del client.

Alla fine del ciclo di vita di un entity bean il container richiama il metodo `unsetEntityContext` che rimuove il bean dal pool.

Come si è detto ogni entity bean ha un suo identificatore unico, che ne costituisce la primary key. Questa può essere implementata da una qualsiasi classe Java serializzabile. Tipicamente questa classe contiene semplicemente delle variabili di istanza corrispondenti alle variabili di istanza della primary key della tabella del database associata. Il client può utilizzare il metodo `FindByPrimaryKey`, che deve essere implementato nella classe del bean, passando come argomento una particolare istanza della classe primary key per ottenere l'entity bean specifico corrispondente. Il client può avere a disposizione anche altri metodi cossiddetti *finder* per recuperare una particolare entity bean passando altri valori univoci.

Il legame fra un entity bean e i dati che rappresenta sul database è talmente stretto che una modifica nel primo si riflette in un aggiornamento dei dati. Se il database da utilizzare per la memorizzazione dei dati è di tipo relazionale, però, si può creare un problema nel mappaggio dei dati con le variabili del bean. Infatti la logica a oggetti del linguaggio Java consente di realizzare strutture dati di complessità elevata rendendo difficile l'operazione di conversione nel formato accettato da un database relazionale. Il container ha a disposizione due metodi, `ejbLoad` e `ejbStore`, per sincronizzare i dati memorizzati nel database con quelli incapsulati nel bean. Tipicamente ogni invocazione di un metodo associato ad una transazione prevede prima l'invocazione di `ejbLoad` , poi l'esecuzione del metodo ed infine l'invocazione di `ejbStore`.

2.4.2 Un esempio di EJB

Si riporta a titolo di esempio il codice di uno stateless session bean che realizza un euroconvertitore.

Codice dell'EJB

Si è supposto di includere le classi Java del'EJB in un package di nome "ejb".

Per prima cosa si scrive il codice della remote interface che contiene la dichiarazione dei metodi che potranno essere invocati dal client.

```
Package ejb;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Euroconvertitore extends EJBObject {

     public double daEuroAlire(double euro) throws
RemoteException;

     public double daLireAeuro(double lire) throws
RemoteException;

}
```

- Remote interface dell'EJB -

Il passo successivo è quello di scrivere il codice della home interface che contiene i metodi di creazione e rimozione del bean. Essendo stateless non occorre alcun metodo di rimozione, ma solo di creazione.

```
Package ejb;

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
```

```
import javax.ejb.EJBHome;

public interface EuroconvertitoreHome extends EJBHome {

   Euroconvertitore create() throws RemoteException,
CreateException;

}
```

- Home interface dell'EJB -

Poi si scrive il codice della classe del bean che implementa i metodi della remote interface.

```
Package ejb;

import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class EuroconvertitoreEJB implements SessionBean {

      public double daEuroAlire(double euro) {
            return euro * 1936.2700;
      }

      public double daLireAeuro(double lire) {
            return lire * 0.000516457;
      }

      public EuroconvertitoreEJB() {}
      public void ejbCreate() {}
      public void ejbRemove() {}
      public void ejbActivate() {}
      public void ejbPassivate() {}
      public void setSessionContext(SessionContext sc) {}

}
```

- Classe dell'EJB -

Il deployment descriptor dell'EJB

Una volta compilate le interfacce e la classe del bean, è necessario farne il deployment impacchettandole in un file EJB-JAR assieme al deployment descriptor.

Il deployment descriptor è un file XML, di cui si è già accennato e che deve chiamarsi ejb-jar.xml, che contiene informazioni specifiche sul bean e consente all'EJB container di istanziarlo e mettere a sua disposizione i servizi per cui viene configurato. Le specifiche EJB forniscono il DTD che deve essere seguito per scrivere il deployment descriptor.

Per una trattazione generale sui deployment descriptors si veda il paragrafo 2.6.

Si riporta qui di seguito la configurazione minima dell'EJB dell'euroconvertitore da riportare nel deployment descriptor.

```
<?xml version="1.0" encoding="UTF-8"?>

<ejb-jar>

<description>EJB che consente di eseguire una conversione
di valuta da euro a lire e viceversa</description>

<display-name>Euroconvertitore</display-name>

<enterprise-beans>
<session>

      <ejb-name>ejb.Euroconvertitore</ejb-name>
      <home>ejb.EuroconvertitoreHome</home>
      <remote>ejb.Euroconvertitore</remote>
      <ejb-class>ejb.EuroconvertitoreEJB</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
```

```
</session>
</enterprise-beans>

</ejb-jar>
```

- Deployment descriptor -

Come si nota il file deve sempre cominciare con un tag <ejb-jar> e al suo interno si trova un altro tag <enterprise-beans> che contiene a sua volta un tag <session> e/o un tag <entity> per ogni session bean e/o entity bean contenuto nel file .jar . Nell'esempio il pacchetto conterrà solo un session bean, perciò viene riportato un unico tag <session> che conterrà i parametri di configurazione del bean Euroconvertitore. Tra questi quelli obbligatori sono il nome completo della remote e home interface, il tipo si session bean (stateless o stateful) e il tipo di gestione delle transazioni che si intende utilizzare. E' attraverso quest'ultimo parametro che viene data la possibilità agli sviluppatori di specificare se la gestione delle transazioni deve essere fatta automaticamente dal container (*Container-menaged transaction demarcation*) o esplicitamente inserita nel codice del bean (*Bean-Managed Transaction Demarcation*).

Nell'esempio si è specificato cone transaction-type il valore "Bean" che significa che si lascia allo sviluppatore il compito di definire nel codice i confini delle transazioni.

Specificando invece il valore "Container" si opta per una gestione automatizzata delle transazioni e in tal caso è necessario specificare per ogni metodo del bean un ulteriore transaction-attribute il cui valore indica quale tipo di definizione dei confini delle transazioni si vuole applicata al metodo stesso. I possibili valori di questo attributo sono i seguenti:

1) *Required* : il container garantisce che il metodo sia invocato sempre nel contesto di una transazione JTA. Se il client chiamante è già associato ad una transazione JTA si mantiene lo stesso contesto con meccanismo di propagazione automatica,

altrimenti viene creata una transazione JTA che comincia all'inizio del metodo e fa commit al ritorno, con gestione automatica del rollback nel caso venga lanciata un'eccezione.

2) *RequiresNew* : il container crea sempre una nuova transazione JTA prima dell'invocazione del metodo e fa il commit al ritorno. Se il client chiamante è già associato ad una transazione JTA, questa viene temporaneamente sospesa e ripresa al ritorno.
3) *NotSupported* : il contesto transazionale del client non viene propagato al metodo invocato. Se il client chiamante è già associato ad una transazione JTA, questa viene temporaneamente sospesa e ripresa al ritorno.
4) *Supports* : se il client chiamante è già associato al contesto di una transazione JTA questo viene propagato al metodo, altrimenti il metodo viene eseguito senza associazione ad alcuna transazione.
5) *Mandatory* : il container richiede che il client chiamante sia già associato ad una transazione JTA altrimenti viene lanciata una eccezione.
6) *Never* : il container richiede che il client chiamante non sia già associato ad una transazione JTA altrimenti viene lanciata una eccezione.

Codice del client

Ora si passa ad illustrare i punti più significativi del codice di un client che intenda utilizzare l'EJB appena creato.

La prima cosa che un client deve fare per poter utilizzare un EJB, è recuperare una istanza della sua home interface attraverso un meccanismo che consenta di rintracciare oggetti remoti in esecuzione in spazi di indirizzamento eterogenei. Per fare questo è necessario poter riferire l'oggetto in questione attraverso un nome logico che sia indipendente dallo spazio di indirizzamento remoto.

La J2EE utilizza come standard per consentire una associazione tra risorse e nomi logici le API Java JNDI di cui si è già accennato. Esse forniscono due tipi di servizi:

1) un servizio di *binding* cioè la registrazione di un oggetto tramite un nome logico;
2) un servizio di *lookup* cioè il recupero di un oggetto a partire dal suo nome logico.

Al momento del deployment il container effettua un'operazione di binding dell'home interface del bean sotto il nome logico, specificato nel tag `<ejb-name>` del deployment descriptor, relativo ad un certo *naming context* che la J2EE definisce con il nome "java:comp/env" e che funge da radice nella gerarchia dei nomi.

Il nome logico completo con il quale un client può riferire il bean dell'Euroconvertitore è dunque "java:comp/env/ejb/Euroconvertitore".

Le specifiche EJB raccomandano di ottenere la home interface di un EJB come Object generico attraverso il nome JNDI con un'operazione di lookup, e poi di convertirlo in una istanza della home interface utilizzando il metodo `narrow` della classe `javax.rmi.PortableRemoteObject`.

Si riporta qui di seguito il codice di un semplicissimo client che ottiene una reference alla home interface del EJB Euroconvertitore e esegue i suoi metodi di conversione per alcuni valori stampandone il risultato.

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import Euroconvertitore;
import EuroconvertitoreHome;

public class EuroconvertitoreClient {

      public static void main(String[] args) {
```

```
        try {
            // si ricava il context iniziale
            Context initial = new InitialContext();

            /* poi si ottiene un riferimento alla home
        interface come oggetto generico */
            Object objref =
        initial.lookup("java:comp/env/ejb/Euroconvertitore");

            /* si converte l'oggetto in una istanza della
        home interface */
            EuroconvertitoreHome home =
                 (EuroconvertitoreHome)PortableRemoteObject
            .narrow
                              (objref,
            EuroconvertitoreHome.class);

            Euroconvertitore ec = home.create();
            double amount = ec.daEuroAlire(100.00);
            System.out.println(String.valueOf(amount));
            amount = ec.daLireAeuro(100.00);
            System.out.println(String.valueOf(amount));
            ec.remove();

        } catch (Exception ex) {

            System.err.println("Si e' verificato un errore: ");
            ex.printStackTrace();
        }
}
```

- Un semplice client che utilizza l'EJB -

2.4.3 Web tier

Esistono diverse tecnologie che consentono la generazione di pagine web dinamiche sulla base dei dati provenienti da varie sorgenti.

La prima tecnologia sviluppata era la *CGI (Common Gateway Interface)*, un'interfaccia per mezzo della quale il server web era in grado di lanciare applicazioni residenti sul server stesso.

La J2EE platform raccomanda invece l'utilizzo di JSP e servlet per la generazione di pagine web dinamiche. Tali tecnologie consentono di sviluppare applicazioni web con caratteristiche di portabilità, scalabilità e mantenibilità non raggiungibili attraverso la CGI.

La CGI infatti impone alcune limitazioni:

1) il codice inserito in un CGI script che accede a risorse come il file system o un database deve essere specifico per la piattaforma del server, perciò viene a mancare il vantaggio della portabilità che non consente l'utilizzo di questa tecnologia in ambienti distribuiti dove le applicazioni web devono funzionare su piattaforme diverse;
2) ogni volta che uno script CGI viene invocato deve essere creato un nuovo processo, con conseguente spreco di risorse, lentezza d'esecuzione e limitata scalabilità;
3) le applicazioni che utilizzano CGI mischiano il codice relativo alla presentation-logic e quello relativo alla business logic.

Servlet

Le servlet sono componenti web scritti in linguaggio Java. Tali componenti possono dunque essere supportati da web-server con qualsiasi piattaforma e possono essere portati su altre piattaforme senza necessità di essere ricompilati.

Inoltre le servlet vengono caricate in memoria una sola volta e ogni volta che una HTTP request le richiede, questa viene servita da un thread diverso della stessa JVM, invece che da un nuovo processo, perciò il vantaggio della scalabilità è assicurato senza necessità di hardware aggiuntivo.

Infine uno dei benefici maggiori che merita di essere menzionato è la possibilità di utilizzare API uniformi per mantenere dati visibili attraverso richieste HTTP multiple, con particolare riferimento al concetto di sessione.

A proposito di quest'ultima caratteristica è bene tener presente che esistono quattro ambiti (*scope*) di esistenza, condivisione e visibilità delle sorgenti di informazioni di una applicazione web; ciascuno di questi scope è associato ad un particolare oggetto, detto appunto *scope object*, e le informazioni ad esso associate vengono mantenute come attributi di tale oggetto.

I quattro scope object con la relativa visibilità sono i seguenti:

1) *web context* : rappresenta l'ambito con maggiore visibilità; le informazioni memorizzate come attributi sono visibili a tutti i web component dell'applicazione
2) *session* : rappresenta un ambito associato ad un utente; e informazioni memorizzate come suoi attributi sono visibili a tutti i web component che servono le request di uno stesso client. E' questo il classico esempio del carrello della spesa delle applicazioni e-commerce
3) *request* : rappresenta l'ambito associato ad una stessa request
4) *page* : rappresenta l'ambito associato ad una stessa JSP

Il problema di avere risorse condivise, comporta una adeguata gestione della concorrenza tra i vari accessi. Tra gli scenari possibili di accessi concorrenti non vi sono solo quelli in cui più web components accedono a risorse del web context, o della stessa session, ma anche quelli in cui più thread dello stesso web component accedono a variabili di istanza. Come si è detto infatti il web container crea un nuovo thread per ciascuna request e non è quindi garantita l'atomicità dell'esecuzione di un metodo, a meno di non specificare la clausola synchronized. Un altro sistema per garantire l'atomicità del metodo service (quello che serve di fatto la request) è quello di far implementare alla servlet l'interfaccia `SingleThreadModel`.

E' opportuno dichiarare synchronized tutti i metodi che interagiscono con risorse condivise, siano esse residenti in memoria centrale o secondaria (file system e databases).

Le servlet hanno un ciclo di vita che viene gestito dal web-container. Quando una request è mappata ad una servlet, il container esegue i seguenti passi:

1) carica la classe Java della servlet;
2) ne crea una istanza;
3) inizializza l'istanza invocando il metodo `init`; questo metodo viene richiamato una sola volta all'atto del caricamento della servlet stessa e può essere ridefinito per personalizzare la configurazione iniziale della istanza. Viene inoltre assicurata la sincronizzazione automatica dell'esecuzione di questo metodo in modo da consentirne l'accesso ad un thread per volta;
4) invoca il metodo service passando come parametri un oggetto `HttpRequest` e un oggetto `HttpResponse`.

Se il container necessita di rimuovere una servlet, ne richiama il metodo `destroy`.

Volendo entrare un po' più nei dettagli si riporta qui di seguito un breve esempio di una servlet.

```
Import java.util.*;
Import java.text.*;

public class SimpleServlet extends HttpServlet {


  /* ridefinizione del metodo doGet */

  public void doGet (HttpServletRequest request,
  HttpServletResponse response)
       throws ServletException, IOException {

     PrintWriter out;
```

```
        // istanziamento degli oggetti per data e ora
        String dateformat = "EEEE d MMMM yyyy";
        String timeformat = "H:mm";
        DateFormat df = new SimpleDateFormat(dateformat);
        DateFormat tf = new SimpleDateFormat(timeformat);
        Date datetime = new Date();

        // prima occorre settare il content type
        response.setContentType("text/html");

             // poi costruire la response
        out = response.getWriter();
        out.println("<HTML><HEAD><TITLE>");
        out.println("DATA E ORA");
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>DATA E EORA ATTUALI</H1>");

        out.println(df.format(datetime));
        out.println("<BR>");
        out.println(tf.format(datetime));
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

- Esempio di una servlet che genere una pagina dinamica con data e ora -

Questo è un banalissimo esempio di una servlet che genera una pagina web che contiene la data e l'ora attuali nel momento in cui la pagina viene spedita al client.

Si analizzano ora brevemente le parti più significative del codice.

Una classe Java per essere una servlet deve estendere la classe astratta `HttpServlet` e deve ridefinire almeno un metodo di questa classe tra i seguenti: `doGet`, `doPost`, `doDelete`, `doPut` a seconda del tipo di request HTTP da gestire. In particolare il metodo `doGet` gestisce le request HTTP di tipo GET, cioè quelle che includono eventuali parametri attaccandoli in coda all'URL consentendo quindi un limitato invio di dati, mentre il metodo `doPost` gestisce le request HTTP

di tipo POST che includono eventuali parametri nella request stessa, consentendo invio di un quantitativo di dati molto più ampio. A tutti questi metodi devono essere passati come argomenti la `HttpRequest` ricevuta dal client e la `HttpResponse` da settare ed inviare al client.

Nell'esempio il metodo utilizzato è il `doGet` e la request non contiene parametri. Inoltre si è fatto uso di istanze di classi apposite per ottenere la data attuale opportunamente formattata.

Per costruire la response occorre prima settare i campi dell'intestazione (*header fields*) : tra questi in particolare il campo content-type è quello che definisce il tipo di contenuto del body della response. Nell'esempio il metodo `setContentType` dell'interfaccia `HttpResponse` viene utilizzato per settare il content-type al valore "text/html" che indica che il body della response conterrà del testo in formato HTML, cioè la sorgente di una pagina web che può essere interpretata da un browser.

Per costruire il body della response occorre innanzitutto ottenere un output stream da usare per inviare dati al client. Dal momento che, come si è detto, il contenuto della response dovrà essere interpretato dal browser del client come testo HTML, nell'esempio si è utilizzato il metodo `getWriter` per ottenere un oggetto `PrintWriter` che consente di inviare uno stream di caratteri. Se si fosse utilizzato il metodo `getOutputStream` si sarebbe ottenuto uno oggetto `OutputStream` associato alla response sul quale sarebbe stato possibile inviare uno generico stream di bytes in forma binaria[4].

Come si nota le servlet utilizzano degli statement Java di stampa per l'invio di HTML al client. La pagina viene costruita dal codice dell'applicazione che scrive nello stream di output il codice HTML carattere per carattere.

Il risultato dell'elaborazione di questa servlet è la seguente pagina HTML:

```
<HTML>
```

[4] E' questa la tecnica adottata per eseguire il download dei files nell'applicazione realizzata. Vedi paragrafo 4.4.2 nella sezione dedicata all'implementazione della servlet download.

```
    <HEAD>
        <TITLE>DATA E ORA</TITLE>
    </HEAD>
    <BODY>
        <H1>DATA E ORA ATTUALI</H1>
        lunedi' 4 febbraio 2002<BR>
          15:58
    </BODY>
</HTML>
```

- HTML prodotto dalla servlet precedente -

JSP (Java Server Pages)

Includere l'HTML negli statement di stampa, come avviene nelle servlets, presenta due inconvenienti:

1) un Web designer non riesce ad avere una chiara visione di come sarà l'aspetto della pagina generata prima che la servlet venga eseguita a run-time;
2) se si deve realizzare una modifica nell'aspetto della pagina è molto difficile localizzare la sezione di codice responsabile di tale aspetto.

L'esempio di servlet illustrato precedentemente è molto semplice, per cui non risultano molto evidenti gli inconvenienti di cui sopra. Si immagini però una servlet che generi una pagina HTML molto più elaborata con tantissimi tags innestati tra loro e si avrà così chiara la problematica in questione.

Per ovviare a questi problemi la J2EE raccomanda l'utilizzo di JSP per la generazione di pagine web dinamiche.

Le JSP sono documenti di testo che forniscono una descrizione di come processare una request per creare una response utilizzando un determinato protocollo (l'HTTP è quello di default).

Sarà compito del web-container interpretare nel modo corretto il contenuto di una JSP traducendolo in una servlet che potrà poi essere eseguita dal server per generare la pagina dinamica. Le specifiche servlet e JSP della Sun forniscono le

direttive standard che deve seguire un web-container che supporta le servlet (detto quindi anche servlet-container).

Una JSP è quindi una combinazione di due tipologie di testo:

1) template fissi, che rappresentano il contenuto statico destinato a rimanere tale;
2) espressioni JSP basate sull'utilizzo di diversi paradigmi, che verranno tradotte opportunamente dal web-container.

I template statici possono essere inseriti direttamente nella pagina senza doverli racchiudere in statement di stampa, cosa che verrà fatta automaticamente dal web-container. Inoltre il contenuto statico non si limita al solo HTML, ma può comprendere anche qualsiasi altro formato text-based come l'XML.

Per quanto riguarda le espressioni JSP, queste possono essere riconosciute come tali e opportunamente interpretate dal web-container solo se rientrano nelle seguenti quattro differenti categorie:

1) ***Directive***: sono direttive di carattere generale, indipendenti dal contenuto specifico della pagina, relative alla fase di traduzione/complilazione; Sintassi: `<%@ directive ... %>`.

Le directive sono tre:

a) *page* : definisce diverse proprietà relative alla pagina stessa che vengono comunicate al web-container;
b) *taglib* : dichiara le tag libraries che verranno usate nella pagina (vedere oltre);
c) *include* : comunica al container che nella fase di traduzione deve includere nella pagina il contenuto di un file il cui nome viene specificato nella direttiva stessa;

2) ***Action***: vengono eseguite nella fase di processing e danno origine a codice Java specifico per la loro esecuzione; Sintassi: quella dei tags XML, cioè `<tag attr1="value1" attr2="value2" ... > body </tag>`
3) ***Scripting element***: sono frammenti di codice Java, o in un altro linguaggio specificato e si dividono a loro volta in tre sottocategorie:

a) *Scriptlet*: sono veri e propri statement scritti nello scripting language che danno origine a porzioni di codice Java all'atto della traduzione in servlet; Sintassi: `<% codice %>`

b) *Declaration*: sono dichiarazioni che vengono inserite nella classe Servlet come elementi della classe stessa, senza essere racchiuse in alcun metodo. Possono essere sia variabili di classe che metodi. Se si tratta di variabili, la loro durata e' quella del Servlet quindi sopravvivono e conservano il loro valore nel corso di tutte le esecuzioni dello stesso Servlet. Sintassi: `<%! declaration [declaration] ... %>`

c) *Expression*: contengono un'espressione che segue le regole delle espressioni dello scripting language; l'espressione viene valutata e scritta nella pagina di risposta nella posizione corrispondente a quella dell'espressione JSP. Sintassi: `<%= expression %>`

4) ***Comment***: sono commenti che riguardano la pagina JSP e che vengono eliminati dal web-container nella fase di traduzione; sintassi: `<%-- comment --%>`

Una JSP che realizza la pagina dell'esempio precedente è la seguente:

```
<%@ page import = "java.util.* , java.text.*"
        contentType = "text/html" %>

<%   String dateformat = "EEEE d MMMM yyyy";
     String timeformat = "H:mm";
     DateFormat df = new SimpleDateFormat(dateformat);
     DateFormat tf = new SimpleDateFormat(timeformat);
     Date datetime = new Date();
%>

<HTML>
     <HEAD>
          <TITLE>DATA E ORA</TITLE>
     </HEAD>
     <BODY>
```

```
            <H1>DATA E ORA ATTUALI</H1>
            <%= df.format(datetime) %><BR>
              <%= tf.format(datetime) %>
      </BODY>
</HTML>
```

- JSP che realizza la pagina dinamica con data e ora -

Essa verrà tradotta dal web-container nella servlet precedente la cui esecuzione fornisce esattamente la pagina HTML dell'esempio visto prima.

Si è fatto uso di una Directive page nella quale si sono specificate due proprietà associate alla JSP, la prima mediante l'attributo import al quale è stato assegnato come valore la lista dei package Java da importare, la seconda mediante l'attributo contentType al quale si è assegnato il valore "text/html", cioè il tipo di contenuto da inviare al client come pagina HTML. Quest'ultimo attributo se non specificato viene assunto per default come "text/html" con charset = Latin-1 ad indicare il set di caratteri utilizzato per codificare la pagina.

Come si può notare, per mezzo di una JSP un Web designer può avere subito una immagine chiara di come risulterà la pagina web. Inoltre, essendo le JSP dei documenti di testo, egli può anche utilizzare dei tool per creare e visionare il loro contenuto.

La J2EE raccomanda anche di fare meno uso possibile di elementi scriptlet.

Una JSP con molti scriptlet al suo interno, infatti, può causare alcuni problemi:

1) diventa più difficile da leggere ed è necessaria la conoscenza del linguaggio di scripting per capirne la funzionalità;
2) nel caso in cui alcune funzionalità debbano essere ripetute in più pagine, occorre riportare gli scriptlet ricopiandoli con un copia e incolla, con evidente ridondanza di codice e difficoltà di manutenzione;

Al contrario una JSP con pochi scriptlet, o addirittura senza, non solo evita i problemi sopraelencati, ma consente una ancor più netta seprazione del codice dalla presentazione, favorendo in tal modo la manutenzione e la separazione dei ruoli, e

quindi il parallelismo, tra web designer che può concentrarsi sull'aspetto della presentazione, e il web developer che si occupa dei contenuti trascurando l'interfacciamento grafico.

I due principali meccanismi che consentono di raggiungere questo obiettivo sono i JavaBeans e i Custom Tags.

JavaBeans

I JavaBeans sono delle classi Java facilmente riusabili e composte insieme in una applicazione. Ogni classe Java che segue certe regole di composizione, può essere un JavaBean component.

Tali regole governano le proprietà della classe del JavaBean e i metodi pubblici che ne danno accesso.

Ogni JavaBean, infatti ha una serie di proprietà che possono essere di sola lettura, di sola scrittura o di lettura e scrittura.

Ogni proprietà leggibile deve avere un metodo pubblico getter del tipo

```
PropertyClass getProprty() { ... }
```

e ogni proprietà settabile deve avere un metdo pubblico setter del tipo

```
setProperty ( PropertyClass pc) { ... }
```

Inoltre un JavaBean deve avere un costruttore senza parametri.

Le JSP supportano l'utilizzo di JavaBeans diretto attraverso Actions che possono essere inserite nella pagina:

1) ***jsp:UseBean*** : serve per utilizzare un bean già esistente o crearne una nuova istanza;

i suoi attributi sono:

   a) *id* : definisce il nome di una variabile che identifica il bean nello scope specificato;

b) *scope* : definisce l'ambito di esistenza e di visibilità della variabile il cui nome è definito dall'attributo id. Si tratta dunque di un attributo che specifica a quale scope object andrà associato il bean. I valori ammessi sono i seguenti:
   i) *page* : la variabile è utilizzabile solo all'interno della pagina in cui compare il tag, o di una pagina inclusa staticamente;
   ii) *request* : la variabile è utilizzabile nell'ambito di una singola request;
   iii) *session* : la variabile è utilizzabile nell'ambito di un'intera sessione; la pagina in cui il bean è creato deve contenere una direttiva page con l'attributo session settato a true;
   iv) *application* : la variabile è utilizzabile nell'ambito dell'intera applicazione da tutte le sue pagine JSP.
c) *class* : definisce il nome della classe Java a cui il bean appartiene;
d) *type* : identifica il tipo della variabile il cui nome è definito dall'attributo id, che può così anche non essere necessariamente la classe specificata dall'attributo class, ma anche una superclasse o un'interfaccia implementata dalla classe;

2) ***jsp:getProperty***: inserisce nella pagina il valore di una proprietà del bean;
3) ***jsp:setProperty***: assegna il valore di una o più proprietà del bean

Il body contenuto all'interno dell'Action `jsp:useBean` viene eseguito solo all'atto dell'istanziamento del bean.

L'utilizzo dei JavaBeans è quindi vantaggioso poiché consente di incapsulare del codice Java al suo interno e di inserirlo nella JSP attraverso i tag sopraelencati, invece di immetterlo direttamente nella pagina come scriptlet.

Dalla pagina è possibile manipolare il bean attraverso il settaggio e il ripescaggio delle sue property, ma anche richiamando direttamente da uno scripting element un suo metodo attraverso la variabile identificata col nome specificato nell'attributo id.

Ritornando all'esempio di prima, si può creare il seguente bean:

```
import java.util.*;
import java.text.*;

public class DateTime {

      String dateformat = "EEEE d MMMM yyyy";
      String timeformat = "H:mm";
      DateFormat df = new SimpleDateFormat(dateformat);
      DateFormat tf = new SimpleDateFormat(timeformat);

      public String getDate() {

            Date datetime = new Date();
            return df.format(datetime);

      }

      public String getTime() {

            Date datetime = new Date();
            return tf.format(datetime);

      }

}
```

- JavaBean per incapsulare il codice per visualizzare data e ora -

Poi si può richiamare il bean dalla pagina utilizzando le Actions descritte, perciò la JSP avrà il formato riportato qui di seguito, nel quale si è omessa la Directive pochè non essendo più necessario specificare l'attributo import e avendo assunto il contentType di default :

```
<jsp:useBean id="datetime" class="DateTime" scope="page"
/>

<HTML>
      <HEAD>
```

```
          <TITLE>DATA E ORA</TITLE>
     </HEAD>
     <BODY>
          <H1>DATA E ORA ATTUALI</H1>
          <jsp:getProperty name="datetime"
property="date"/>
          <BR>
            <jsp:getProperty name="datetime"
property="time"/>
     </BODY>
</HTML>
```

- JSP che utilizza il precedente JavaBean per generare data e ora -

In quest'ultima versione della pagina JSP si può apprezzare la differenza con la versione script: facendo uso del JavaBean per incapsulare il codice Java la pagina appare molto più pulita e compatta e il contenuto risulta di facile e intuitiva comprensione. Inoltre tutta la funzionalità racchiusa nel JavaBean può essere utilizzata anche da web designers che, senza necessità di conoscenza del linguaggio Java, possono occuparsi esclusivamente del layout delle pagine in cui dovrà essere inserita la data e l'ora. Qualora si volesse effettuare una qualunque modifica, ad esempio cambiare il formato di visualizzazione della data, sarebbe sufficiente cambiare la stringa `dateformat` e ricompilare il JavaBean per ottenere automaticamente l'aggiornamento di tutte le pagine JSP che lo utilizzano, oppure si potrebbe prevedere un metodo `setFormat` che consenta di personalizzare il formato di pagina in pagina semplicemente settando una property del bean.

Custom Tags

Come si è visto i JavaBeans forniscono un potente strumento per incapsulare funzionalità di presentation logic da inserire in una JSP. Essi però sono manipolabili solo attraverso le loro properties, mentre in taluni casi potrebbe essere utile poter far uso di strumenti in grado di offrire maggiore flessibilità.

Dalla versione 1.1 delle specifiche JSP è stata introdotta un'importante innovazione: la definizione delle *tag extension*, ossia un meccanismo attraverso il quale gli sviluppatori possono estendere le JSP con tags creati ad hoc, i cosiddetti *custom tags*, che, una volta dichiarati e inclusi nella JSP, diventeranno delle action a tutti gli effetti. Possono così essere create delle vere e proprie librerie di nuovi tags (*tag libraies*) con le caratteristiche di riusabilità e portabilità, che forniscono agli web designers potenti strumenti con sintassi XML a loro molto più familiari che non i linguaggi di scripting.

Per implementare un custom tag si deve definire una classe, chiamata genericamente *tag handler*, che implementi una delle tre interfacce `Tag`, `BodyTag` o `IterationTag`, definite nel package `javax.servlet.jsp.tagext`.

L'interfaccia `Tag` viene usata per implementare un tipo di tag empty, cioè non dotato di body, oppure per implementare una action che richiede semplicemente l'esecuzione di operazioni quando viene incontrato il tag iniziale e altre operazioni quando viene incontrato quello finale. Questa interfaccia dunque contiene i metodi di base che sono tipici di tutti i tag handlers, ossia metodi setter per inizializzare il tag con un context e gli eventuali attributi dell'azione, e i metodi che devono essere ridefiniti per implementare la funzionalità vere e proprie da attribuire al custom tag; in particolare questi ultimi sono due: `doStartTag` e `doEndTag`.

Il metodo `doStarTag` deve contenere le operazioni che il container dovrà eseguire nel momento in cui nell'elaborazione della JSP viene incontrato il tag iniziale. Esso restituisce una valore intero che indica al container se e come compiere un processing del body, qualora il tag lo preveda. In particolare esistono due differenti valori di ritorno che indicano due differenti alternative:

1) ignorare il body (utilizzato per i tag empty);
2) valutare il body e inserirne la valutazione direttamente nello stream di output della response (utilizzato ad esempio per inclusioni condizionali).

Il metodo `doEndTag` deve contenere le operazioni che il container dovrà eseguire nel momento in cui nell'elaborazione della JSP viene incontrato il tag finale.

Il suo valore di ritorno è un intero che indica al container se il resto della pagina deve essere valutato o meno.

L'interfaccia IterationTag è un'estensione della interfaccia Tag che fornisce un ulteriore metodo, `doAfterBody`, invocato dal container per la rivalutazione del body del tag, qualora si voglia implementare un tag iterativo. Esso infatti restituisce un intero che indica sel il body deve essere rivalutato ancora o se deve essere richiamato il metodo `doEndTag`.

L'interfaccia `BodyTag` è un'estensione di `IterationTag` che contiene ulteriori due metodi che devono essere ridefiniti qualora si richieda una gestione più complessa del body del tag. Tali metodi sono: `setBodyContent`, `doInitBody`. Il primo è chiamato dal container prima della valutazione del body, solo nel caso questo abbia luogo e assegna al tag un oggetto `BodyContent` che funge da buffer temporaneo per l'output del body. Il secondo viene chiamato dopo il primo soltanto alla prima valutazione del body (e non alle eventuali successive iterazioni).

Per l'interfaccia `BodyTag` il metodo doStartTag può ritornare, come avveniva per l'interfaccia Tag, un valore che indica al container di ignorare il body, mentre l'altra possibilità prevista è quella di valutare il body e inserirlo nel `BodyContent`, invece di mandarlo direttamente sullo stream di output.

Esistono anche delle classi predefinite che implementano le interfacce di cui sopra, e che possono essere utilizzate direttamente come tag handler, al limite ridefinendo solo alcuni metodi strettamente necessari. Esse sono `TagSupport` e `BodyTagSupport`.

All'interno di una JSP è possibile importare più librerie di tag specificandole attraverso la directive taglib. Ciò viene fatto riportando un attributo della directive il cui valore è un URI univocamente associato ad una tag library, e un ulteriore attributo il cui valore è il prefisso che verrà utilizzato per identificare i custom tags della tag library quando verranno inseriti nella JSP.

Nel deployment descriptor web.xml della applicazione web in cui la tag library viene utilizzata deve essere riportato un mapping tra l'URI della tag library e un percorso in cui si trova un particolare file XML detto *TLD (Tag Library Descriptor).*

```
<taglib>
      <taglib-uri>
            http://sparc20.ing.unimo.it:8063/md/taglib
      </taglib-uri>

      <taglib-location>
            /WEB-INF/taglib/tagdescriptor.tld
      </taglib-location>

</taglib>
```

- Esempio della sezione del deployment descriptor che specifica una taglibrary -

Il TLD è un file di configurazione in formato XML, che fornisce al container le informazioni necessarie per poter utilizzare correttamente la libreria. Esso riporta per, ciascuna action della tag library, il nome del tag, il nome della classe del tag handler, informazioni su tutte le eventuali variabili di scripting create dall'action e informazioni sugli eventuali attributi dell'action.

Questo file va inserito in un percorso della applicazione web che deve corrispondere a quello riportato nel tag `<taglib-location>`. Inoltre occorre inserire nel percorso in cui tipicamente vengono riportate le libreie di classi utilizzate dalla applicazione web, tutte le classi utilizzate nella tag library. Solitamente queste classi si trovano impacchettate in un file JAR da inserire nella directory WEB-INF/lib.

Il DTD del file TLD è riportato nelle specifiche JSP dalla versione 1.1 in poi.

2.5 Client tier

Costituisce il front-end più o meno complesso che risiede e viene eseguito sul client che accede alla applicazione. Esso inoltra le richieste al server per conto dell'utente e ne presenta a quest'ultimo i risultati.

La J2EE supporta vari tipi di client che possono connettersi sia all'interno del firewall aziendale che da fuori attraverso il World Wide Web.

Un client può comunicare con, e usare i servizi forniti da, uno o più tiers dell'applicazione enterprise.

A seconda del tier cui si collega, si possono distinguere tre tipologie principali di client: il *Web-client* l'*EJB-client* e l'*EIS-client*

2.5.1 Web-client

Il web client solitamente viene eseguito all'interno di un browser e utilizza i servizi di quest'ultimo per interpretare il contenuto delle informazioni provenienti dal server. L'interfaccia utente è quindi generata a lato server dal Web tier e comunicata via HTML.

Il web-browser è l'unico requisito di ambiente richiesto per questa tipologia di client, che, per questo motivo, viene anche detto "thin client" (client sottile).

A volte è utile fornire direttamente al client alcune funzionalità che aiutano l'utente a meglio interagire con i servizzi del server. In tal caso oltre alle pagine HTML possono venire inviati anche dei Java Applets, cioè delle classi java che vengono scaricate assieme alla pagina HTML che le riferisce ed esegue sul client dall'ambiente di runtime java del browser. Naturalmente le applets sono soggette a particolari restrizioni per garantire la sicurezza e la protezione del client.

La piattaforma J2EE fornisce un supporto speciale per il downolading e l'installazione automatica di una specifica JVM sul client.

I browser spesso supportano anche altri embedded components, come i plug-ins di Netscape e gli ActiveX di Internet Explorer che possono essere scaricati.

I Web-client utilizzano i protocolli di trasmissione HTTP e HTTPS che forniscono i seguenti vantaggi:

- sono diffusissimi: quasi tutti i computer attualmente hanno un browser che supporta l'HTTP;
- presentano il carattere della robustezza e semplicità;
- passano attraverso i fire-wall.

Tali protocolli presentano però allo stesso tempo anche alcuni svantaggi:

- sono stateless: l'HTTP è basato su un modello request-response nel quale a ciascuna richiesta fa corrispondere una e una sola risposta senza mantenere informazioni sulle precedenti richieste e risposte;
- sono non-transazionali: l'HTTP non supporta la possibilità di definire transazioni attraverso richieste multiple.

2.5.2 EJB-client

L'EJB-client è un application client che interagisce con l'EJB tier. Esso consente all'utente interazioni col server molto più potenti di un semplice Web-client grazie alla presenza di una interfaccia utente, tipicamente un programma *GUI (Graphic User Interface)*, più complessa che deve quindi essere installata sul client stesso (il cui ambiente è solitamente un desktop computer).

Ciò presenta anche il vantaggio di avere una distribuzione maggiore del carico di lavoro tra client e server. In particolare il compito di generare l'interfaccia grafica è riservato al client, mentre al server viene lasciato il compito di implementare la sola business-logic. Questo è particolarmente utile per applicazioni con specifici requisiti grafici, per le quali sarebbe troppo gravoso lasciare al server tutta l'elaborazione.

Sono disponibili diversi servizi del middle-tier a questo tipo di client: JNDI, JMS, e JDBC.

Per poter far uso di tali servizi il client deve essere eseguito in un container che tipicamente è molto più semplice dei container del middle-tier. Si tratta infatti di una libreria che viene distribuita assieme al programma client. Essa è specifica per il particolare J2EE service provider, che generalmente provvede a fornirla sottoforma di pacchetto JAR ed un deployment descriptor che definisce gli EJB e le altre risorse esterne alle quali il client può accedere.

Non è richiesto che gli EJB-client supportino le transazioni (se non attraverso gli srumenti forniti dalle API JDBC), perciò solitamente la loro gestione avviene attraverso gli EJB.

Per quanto riguarda la sicurezza esistono particolari restrizioni, in quanto secondo la J2EE platform l'EJB-client deve sempre essere autenticato per poter accedere al middle-tier. Le tecniche di autenticazione sono fornite dal client container e non sono direttamente controllabili dal client stesso.

Il protocollo di trasmissione utilizzato da questi client è l'RMI-IIOP, che consente l'accesso ad oggetti Java definiti usando l'interfaccia RMI (Remote Model Interface) attraverso il protocollo IIOP(). Tale protocollo non è solitamente in grado di passare attraverso un fire-wall, se non con particolari setup aggiuntivi, e questo limita fortemente l'utilizzo di Internet da parte di questi client.

2.5.3 EIS-client

L'EIS-client accede direttamente alle risorse dell'Information System e si assume la responsabilità di mantenere il rispetto dei vincoli e regole della business logic della applicazione. Esso può utilizzare le API JDBC per accedere ai database relazionali, ma attualmente la J2EE non ha ancora definito uno standard implementativo.

Per il fatto che si deve occupare completamente sia della interfaccia utente che della business logic applicativa, questo tipo di client è scarsamente utilizzato e il suo utilizzo dovrebbe essere riservato a compiti di gestione e mantenimento per i quali è richiesta una interfaccia minima o non esistente, come ad esempio un stand-alone

program di manutenzione delle tabelle di database relazionali da lanciare durante la notte.

2.5.4 Problematiche di implementazione

Nell'implementare una applicazione J2EE, è necessario compiere una attenta analisi preliminare riguardo le tipologie di client che potranno accedervi al fine di effettuare le corrette scelte progettuali.

Il primo fattore da considerare è l'ambiente di esecuzione del client, in altre parole stabilire se il client deve essere eseguito in un particolare ambiente con caratteristice dipendenti dalla piattaforma, oppure in un ambiente platform-indipendent. In quest'ultimo caso si potrà scegliere tra due approcci: utilizzare una tipologia di client browser-based dove il browser si occupa di gestire le differenze tra le varie piattaforme, oppure utilizzare un application client scritto in linguaggio Java che assicura l'indipendenza dalla piattaforma, ma richiede l'installazione di una JVM sul client.

Altro fattore da tener presente è il grado di funzionalità che deve avere l'interfaccia utente. Maggior funzionalità viene mantenuta sul client (più vicina all'utente), migliore sarà la percezione della qualità del servizio che l'utente avrà. D'altra parte maggior funzionalità viene lasciata al server, più facile sarà distribuire e gestire l'applicazione.

Un altro fattore di grande importanza da valutare è il tipo di rete a cui il client è collegato. Una rete intranet consente generalmente connessioni veloci e meno variabilità rispetto alla rete Internet. Le applicazioni sviluppate per Internet tipicamente richiedono invece che il client possa funzionare con connessioni lente e con minime risorse di ambiente. Inoltre la rete Internet e le reti intranet hanno differenti livelli di sicurezza. Ai client che si interfacciano con una rete intranet, infatti, non è richiesto di dover interagire con un fire-wall e possono avere lo stesso security domain del server. La presenza di un fire-wall limita la scelta del protocollo

di trasmissione che può essere utilizzato: sul World Wide Web la maggior parte dei fire-wall consente l'accesso ai protocolli HTTP e HTTPS.

2.6 Packaging e deployment

La J2EE platform consente agli sviluppatori di creare differenti parti delle loro applicazioni sotto forma di componenti riusabili. Questi componenti possono essere poi raggruppati in insiemi, detti *moduli*, a seconda di caratteristiche o funzionalità comuni. Il processo di assemblaggio dei componenti in moduli e dei moduli in enterprise applications è detto *packaging*.

Il processo di installazione e personalizzazione di una applicazione in un certo ambiente operativo è detto *deployment*. Affinchè sia possibile personalizzare una applicazione, i suoi componenti devono poter essere configurabili mediante un meccanismo standard.

La J2EE fornisce strumenti standard per semplificare questi processi di packaging e deployment. Essa usa file JAR come package standard per i moduli e le applicazioni, e dei files XML-based per configurare questi moduli detti *deployment descriptors*.

2.6.1 Ruoli e compiti

I processi di packaging e deployment coinvonlgono tre differenti ruoli identificabili in tre figure professionali non necessariamente svolte da persone diverse:

1) *Application component providers* : hanno il compito di sviluppare gli Enterprise Java Beans, le pagine HTML e JSP, e tutte le ulteriori classi associate. Essi forniscono tutte le informazioni strutturali del deployment descriptor per ciascun componente, che includono, come già visto, l'indicazione della home e remote interface e della classe degli EJB, il meccanismo di persistenza usato, e il tipo di

risorse che questi componenti utilizzano. Senza l'ausilio dei deployment descriptors tutte queste informazioni sarebbero da includere nel codice dei componenti riducendo drasticamente la loro flessibilità e riusabilità.

2) *Application assemblers* : forniscono informazioni relative all'applicazione nel suo complesso. Tipicamente ad essi spetta il compito di mappare le varie servlet sui diversi URL, definire le pagine di errore da usare, i vincoli di sicurezza dei vari componenti, ecc.
3) *Deployers* : sono responsabili del deployment dei componenti in un particolare ambiente operativo. Ad essi spetta il compito di installare i diversi componenti sul J2EE server (o sui J2EE servers) fornendo le informazioni aggiuntive specifiche per l'ambiente operativo. Ad esempio essi si occupano di mappare gli utenti e gli accounts esistenti, con i security roles definiti dall'Application assembler.

2.6.2 Moduli J2EE

Un applicazione J2EE è impacchettata in un file Enterprise ARchive (EAR), un file standard Java JAR con estensione .ear. L'obiettivo di questo meccanismo di packaging è di fornire una unità di deployment che sia portabile.

Un file EAR contiene uno o più moduli J2EE e un *J2EE application deployment descriptor*, che contiene indicazioni sui moduli stessi.

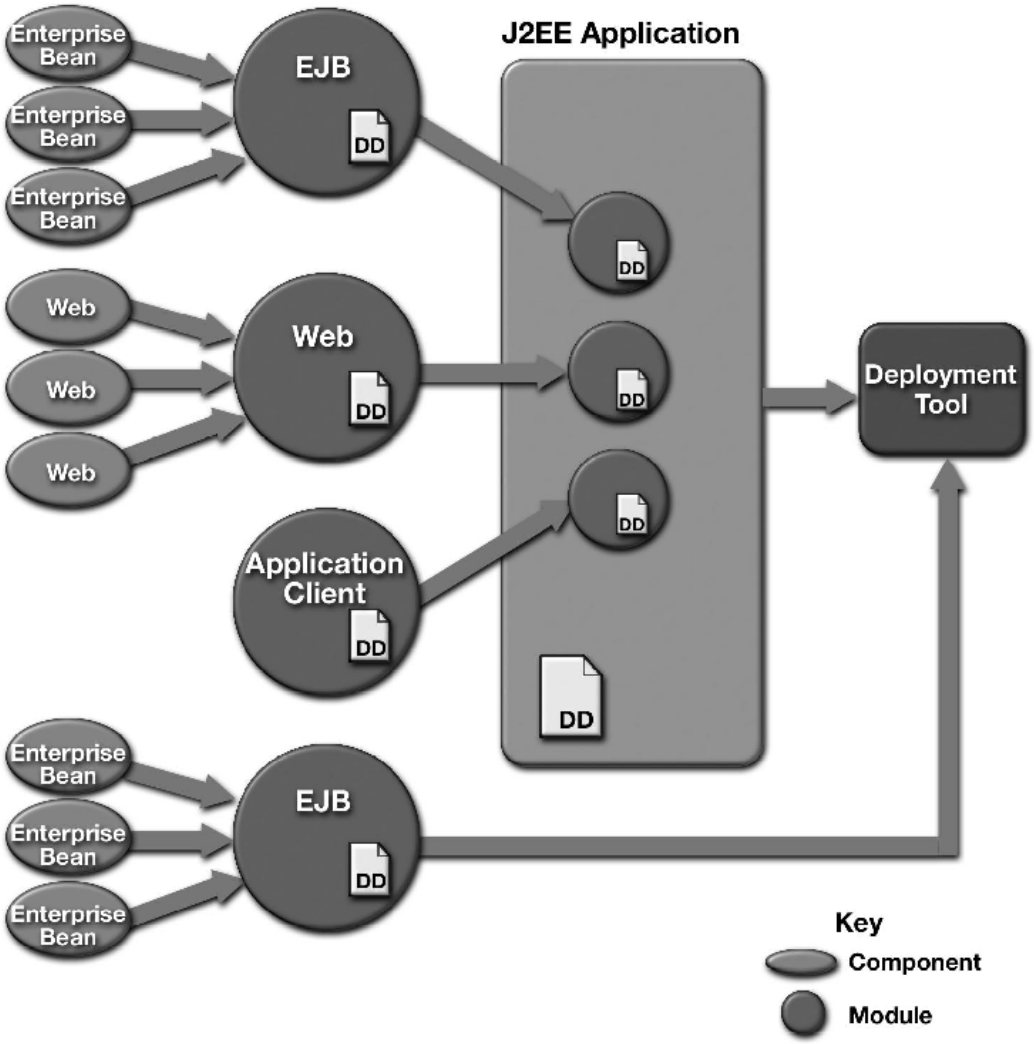


fig 2.6 - Packaging dei moduli J2EE -

Moduli EJB

Un modulo EJB è l'unità più piccola di EJB della quale si può fare il deployment. Tale modulo è impacchettato in un file EJB JAR, cioè un file standard Java JAR con estensione .jar che contiene:

1) le classi Java degli EJB e le loro remote e home interface. Se si tratta di un entity bean deve essere compresa anche la classe della primary key;

2) tutte le altre classi Java dalle quali dipendono gli EJB che non siano già comprese nella J2EE platform;
3) un EJB deployment descriptor, tipicamente chiamato ejb-jar.xml, del quale si è già mostrato il contenuto minimo nell'esempio dell'Euroconvertitore, che deve essere contenuto in una particolare directory di nome META-INF.

E' da notare che un EJB JAR file differisce da un normale JAR file perché nel primo è contenuto anche il deployment descriptor.

Moduli Web

Un modulo Web è l'unità più piccola di risorse Web della quale si può fare il deployment. Tale modulo è impacchettato in un file Web ARchive (WAR), un file standard Java JAR con estensione .war che contiene:

1) le classi Java delle servlet e le classi dalle quali queste dipendono, eventualmente impacchettate a loro volta in un normale file JAR;
2) le pagine JSP e le classi Java dalle quali dipendono;
3) documenti statici, come ad esempio le pagine HTML, le immagini, i file sonori, ecc. ;
4) le applets;
5) un Web deployment descriptor, tipicamente chiamato web.xml e contenuto in una speciale directory chiamata WEB-INF.

Moduli Application client

Un modulo application client è impacchettato in un file JAR con estensione .jar e contiene:

1) le classi Java che implementano il client;
2) un Application client deploynment descriptor che descrive gli EJB e le risorse esterne referenziate dall'applicazione.

La J2EE non specifica tools per effettuare il deployment di un application client, o meccanismi per installarlo. Alcune piattaforme J2EE sofisticate possono consentire il deployment dell'application client direttamente nel J2EE server e metterlo così automaticamente a disposizione di alcuni clients intranet. Altre piattaforme J2EE possono invece richiedere che il deployment dell'application client sia manualmente effettuato su ciascuna macchina client.

2.7 Connection pooling

Quando un componente intende accedere a risorse di un database deve stabilire una connessione. I tempi di attesa per stabilire una connessione sono molto onerosi e possono compromettere le prestazioni di un middle tier server.

Per ovviare a questo inconveniente la J2EE fornisce supporto per il *connection pooling*, cioè si occupa di mantenere e gestire una cache di connessioni al database, così da consentirne il riuso.

Questo supporto viene fornito dall'EJB container in modo del tutto trasparente agli sviluppatori che nel codice dei componenti richiedono una connessione e la richiudono una volta terminata. Sarà compito del container assegnare al componente una delle connessioni già aperte e disponibili nella cache, apporvi un lock in modo che non sia utilizzabile da altri componenti, e liberarla quando il componente ne ha terminato l'utilizzo.

Le API JDBC forniscono una interfaccia standard chiamata `DataSource` per ottenere una connessione del pool. Ogni EJB container vendor deve fornire una implementazione di questa classe che si adatti al particolare tipo di pool utilizzato e faccia uso di uno o più algoritmi di caching per gestire l'assegnamento delle connessioni. Le istanze della classe che implementa questa interfaccia possono essere ottenute attraverso un nome logico utilizzando le solite API JNDI. In tal modo si garantisce nel codice anche un livello di astrazione rispetto al driver JDBC utilizzato e quindi rispetto al DBMS sottostante.

Nel codice dei componenti sarà quindi sufficiente effettuare un'operazione di lookup al nome logico JNDI corrispondente ad una particolare sorgente JDBC e ottenere così la classe che implementa l'interfaccia `DataSource`. Per ottenere la connessione vera e propria si utilizzerà il metodo `getConnection` dell'interfaccia che restituisce un normale oggetto `Connection` che riferisce ad una effettiva connessione aperta del pool e potrà essere utilizzato normalmente. Una volta terminato il suo utilizzo sarà sufficiente chiudere la connessione con il metodo `close` dell'oggetto `Connection` per restituirla al pool.

L'implementazione dell'interfaccia `DataSource` fornita dal EJB container vendor utilizza altre interfacce JDBC, che devono anch'esse essere implementate dal vendor, di nome `PooledConnection` e `ConnectionPoolDataSource` e che sono gestite dal container in modo trasparente. In particolare quando il componente invoca il metodo `DataSource.getConnection`, l'EJB container effettua a sua volta una operazione di lookup nel connection pool per vedere se esiste una istanza di `PooledConnection` che può essere usata. Se esiste, questa viene ritornata al container e viene posto il lock. Se non esiste viene utilizzato un oggetto `ConnectionPoolDataSource` per ottenere una nuova istanza di `PooledConnection`. Quando il container ha ottenuto una istanza di `PooledConnection` chiama il metodo `PooledConnection.getConnection` che restituisce un normale oggetto `Connection` che viene ritornato al componente.

Quando nel codice del componente viene chiamato il normale metodo `Connection.close` viene lanciato un evento che il container cattura e, invece di chiudere fisicamente la connessione, si limita a togliere il lock precedentemente posto.

```
// ottenere il context JNDI iniziale
Context initctx = new InitialContext();

// ottenere l'implementazione del DataSource
```

```
DataSource ds =
(DataSource)initctx.lookup("java:comp/env/jdbc/MyDatabase
");

// ottenere una connessione dal pool
Connection con = ds.getConnection();

// utilizzo della connessione
// ...

// restituzione della connessione al pool
con.close();
```

- Frammento di codice per ottenere una connessione dal pool -

2.8 EJB-Centric vs Web-Centric

La J2EE non pone alcun vincolo sul tipo di approccio da utilizzare per lo sviluppo di una applicazione web multilivello.

Il classico approccio Web-centric, ancora molto diffuso per piccole applicazioni, prevede che sia la presentation logic che la business logic risiedano sul Web tier e che questo si connetta direttamente al EIS tier tramite JDBC.

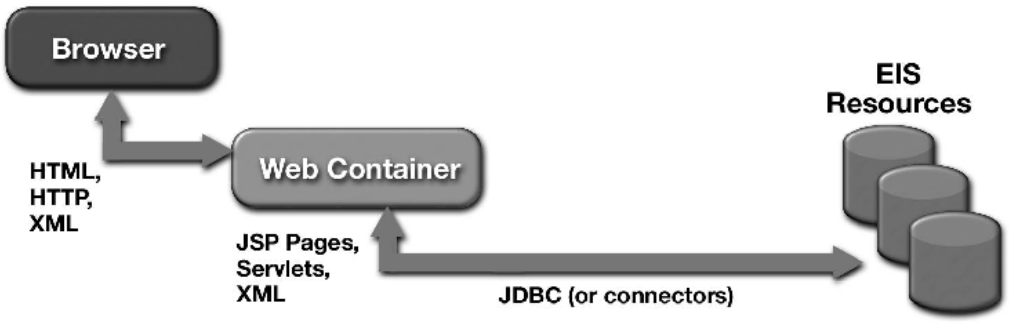


fig 2.7 - Approccio 3-tier Web-centric -

In questo approccio il Web-tier è responsabile di quasi tutta la funzionalità dell'applicazione. Infatti, oltre a processare le request del client per generare una opportuna response, attraverso la produzione delle pagine web dinamiche, il Web-tier deve anche occuparsi di implementare tutte le funzionalità di aggiornamento dei dati nell'EIS-tier. Come conseguenza si ha che il Web-tier in questo tipo di approccio risulta essere notevolmente appesantito, tendendo a diminuire la scalabilità dell'applicazione.

Nell'approccio EJB-centric, come si è detto, tutta la business logic è lasciata all'EJB tier ed è questo l'unico livello direttamente collegato all'EIS tier.

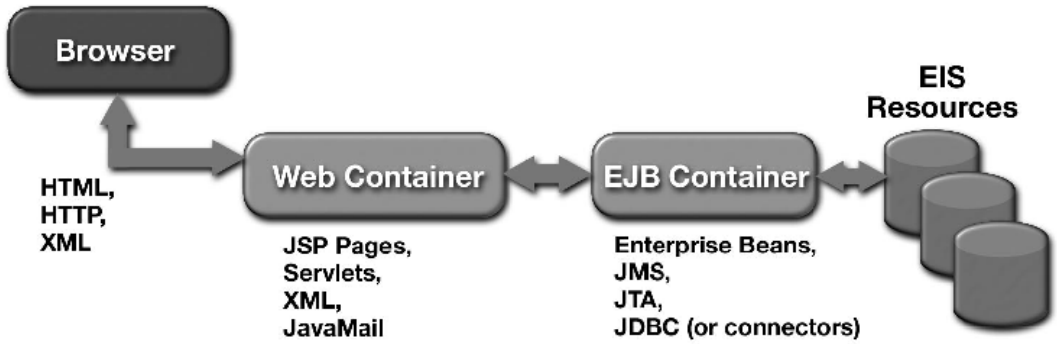


fig 2.8 - Approccio 3-tier EJB-centric -

Il vantaggio principale di quest'ultimo approccio è costituito dalla possibilità di sfruttare le funzionalità messe a disposizione dall'EJB-container. Esso, come si è detto, infatti fornisce supporto per la gestione delle transazioni e per il connection pooling garantendo alti livelli di scalabilità, ideali per applicazioni per le quali si preveda aumento di dimensioni. Inoltre questo approccio è particolarmente adatto per applicazioni distribuite grazie alla possibilità di suddividere Web tier e EJB tier su diversi hosts.

2.9 Sicurezza

In questa sezione viene introdotto il meccanismo per gestire la sicurezza in una applicazione J2EE sia dal punto di vista dell'EJB container sia dal punto di vista del Web container.

Le risorse alle quali un client può accedere possono essere soggette a vincoli di sicurezza, nel qual caso il client necessita di un meccanismo di autenticazione, cioè un criterio che consenta di identificarlo in modo certo. Il client autenticato potrà quindi accedere alle risorse protette solo se autorizzato.

Per controllare gli accessi alle risorse protette si utilizza un meccanismo di mapping degli *users* in *roles*[5].

La J2EE platform fornisce supporto sia ad un tipo di gestione della sicurezza *dichiarativo*, sia *programmatico*. Il primo viene gestito esclusivamente attraverso i deployment descriptors senza includere nulla nel codice dei componenti, mentre il secondo prevede l'utilizzo di apposite API all'interno del codice dei componenti. Per quest'ultimo tipo sono particolarmente utili il metodo `isCallerInRole` dell'interfaccia `EJBContext` utilizzato dagli EJB, e il metodo `isUserInRole` dell'interfaccia `HttpServletRequest` utilizzato dai Web component, entrambi in grado di restituire un valore booleano che indica se l'utente è appartenente al role specificato come parametro.

La trattazione seguente riguarda invece il tipo di gestione dichiarativa della sicurezza.

2.9.1 Sicurezza nel Web tier

Quando un utente tenta di accedere a risorse Web protette, il Web container attiva il meccanismo di autenticazione definito nel deployment descriptor web.xml per quella risorsa.

[5] I roles sono raggruppamenti logici di utenti definiti dall'Application Component Provider o dall'Assembler. Il Deployer mappa i roles agli users.

Un J2EE Web container fornisce supporto per tre tipi di meccanismo di autenticazione:

1) basic authentication : il Web server provvede all'autenticazione del client per mezzo di uno username e una password che questo deve fornire;
2) form-based authentication : è sostanzialmente identica alla basic, ma consente di personalizzare la pagina di autenticazione presentata dal browser al client e anche la pagina di errore nel caso di accesso negato;
3) mutual authentication : sia il client che il server devono autenticasi a vicenda attraverso dei certificati di autenticazione e utilizzano un canale di comunicazione protetto dal *SSL (Secure Socket Layer)*. Questo è il meccanismo più sicuro tra i tre proprio grazie all'utilizzo di SSL e della mutua autenticazione.

Il meccanismo di autenticazione utilizzato deve essere specificato attraverso il tag `<login-config>` del deployment descriptor:

```
<login-config>
      <auth-method>FORM</auth-method>
      <form-login-config>
            <form-login-page>login.jsp</form-login-page>
            <form-error-page>error.jsp</form-error-page>
      </form-login-config>
</login-config>
```

- Configurazione di autenticazione form-based -

Nel deployment descriptor è possibile specificare, per ogni risorsa protetta, i roles che possono accedervi, definendo i cosiddetti *security constraint*. Se uno user che tenta di accedere a tali risorse non appartiene ad uno dei role specificati nel security constraint, allora l'accesso viene negato.

Qui di seguito viene riportato il frammento del deployment descriptor che consente l'accesso ad una risorsa denominata placeorder e mappata all'URL

/control/placeorder , mediante request di tipo GET e POST ai soli users che rientrano nel role customer:

```
<security-constraint>
      <web-resource-collection>
            <web-resource-name>placeorder</web-resource-
      name>
            <url-pattern>/control/placeorder</url-pattern>
            <http-method>POST</http-method>
            <http-method>GET</http-method>
      </web-resource-collection>
      <auth-constraint>
            <role-name>customer</role-name>
      </auth-constraint>
</security-constraint>
```

- Controllo degli accessi a risorse Web protette -

2.9.2 Sicurezza nel EJB tier

Il tipico meccanismo di protezione degli accessi agli enterprise bean è quello in cui l'EJB container si "affida" al Web container che si fa garante dell'identità dell'user che accede a componenti Web protetti.

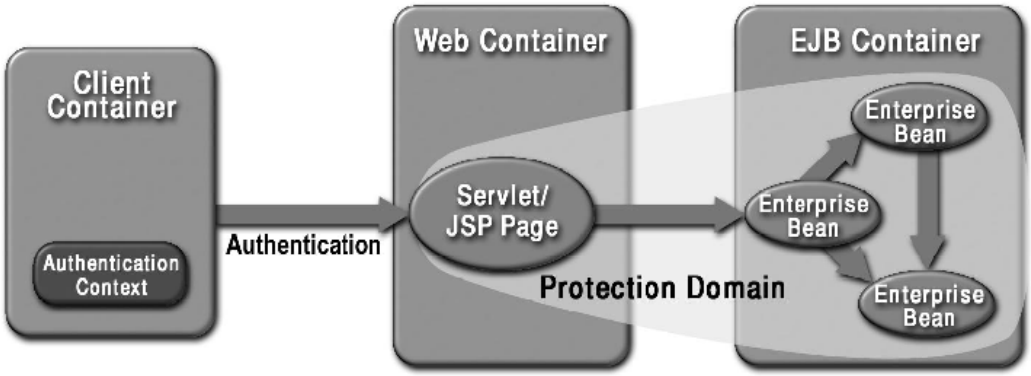


fig 2.9 - Tipica configurazione di autorizzazione in una applicazione J2EE -

La J2EE consente però di definire anche un livello di protezione degli EJB per client stand alone che si connettono attraverso RMI-IIOP.

Nel deployment descriptor ejb-jar.xml è possibile definire, per ogni role, l'insieme dei metodi degli enterprise bean contenuti nel modulo ai quali un utente con quel role può accedere.

Prima di fare questo è però necessario definire tutti i roles utilizzati, con una eventuale descrizione, mediante i tag <security-role> del deployment descriptor:

```
<assemply-descriptor>
      <security-role>
            <description>
            Questo role si riferisce all'amministratore di
sistema
            </description>
            <role-name>admin</role-name>
      </security-role>
      <security-role>
            <description>
            Questo role si riferisce ad un cliente
            </description>
            <role-name>customer</role-name>
      </security-role>
</assemply-descriptor>
```

Nel frammento di deployment descriptor che segue viene realizzato il seguente security constraint: tutti i metodi della remote e home interface del EJB "theOrder" possono essere acceduti dai soli user con role "admin", tranne il metodo getDetails che può essere acceduto anche da user con role "customer"; inoltre i metodi della home e remote interface del EJB "Customers" possono anch'essi essere acceduti dai soli users con role "admin":

```
<method-permission>
      <role-name>admin</role-name>
      <method>
            <ejb-name>TheOrder</ejb-name>
            <method-name>*</method-name>
      </method>
      <method>
            <ejb-name>Customers</ejb-name>
```

```
            <method-name>*</method-name>
      </method>
</method-permission>

<method-permission>
      <role-name>customer</role-name>
      <method>
            <ejb-name>TheOrder</ejb-name>
            <method-name>getDetails</method-name>
      </method>
</method-permission>
```

- Controllo degli accessi a EJB protetti -

Capitolo 3

Implementazione della piattaforma J2EE : JBoss e Tomcat

In questo capitolo viene introdotta la particolare implementazione della piattaforma J2EE utilizzata.

Come Web container si è utilizzata la versione 4.0 di *Tomcat* [TOMCAT] uno dei più diffusi servlet e JSP container, mentre come EJB container è stato utilizzato *JBoss* 2.4.3 [JBOSS].

Le motivazioni che hanno spinto verso questa scelta sono legate principalmente ai vantaggi dell'accopiata JBoss-Tomcat rispetto al J2EE SDK messo a disposizione freeware dalla Sun:

1) i requisiti minimi per JBoss sono di soli 64 Mbyte di RAM e pochi Mbyte di hard disk, contro i 128 Mbyte di RAM e 31 Mbyte di hard disk richiesti dal server J2EE;
2) JBoss ha velocità di avviamento dieci volte superiore ad un server J2EE;
3) JBoss fornisce supporto per il cosiddetto "hot deployment", cioè semplifica notevolmente la procedura di deployment. Il deployment di un bean può infatti essere fatto semplicemente copiando il file JAR che lo contiene in un'opportuna sottodirectory del percorso in cui è installato JBoss; se questo viene fatto quando il bean è già stato caricato, esso viene automaticamente scaricato dal server a favore della nuova versione;

4) JBoss contiene già al suo interno un SQL database-server (Hypersonic) per gestire gli entity beans, che viene automaticamente avviato all'avvio del server, mentre il server J2EE contiene una versione di CloudScape SQL che deve essere avviato separatamente.

3.1 Il Web-container Tomcat

Tomcat è un servlet container che offre supporto per lo standard Servlet 2.2 e per lo standard JSP 1.2. Dalla release 4.0 in poi esso fornisce supporto anche per la nuova versione 2.3 delle specifiche Servlet.

Tomcat è scritto interamente in Java ed è messo a disposizione on line freeware open source dal progetto Jakarta [JAKARTA] .

3.1.1 Configurazione e installazione di Tomcat stand alone

In un approccio Web-centric, come si è già detto, nel middle tier viene implementato solo un Web container. E' questo il caso in cui è necessario installare e configurare Tomcat come web server stand alone, cioè in modo da poter funzionare in una VM dedicata.

Un altro caso in cui è necessario configurare Tomcat stand alone è quello in cui Web container e EJB container sono dislocati su differenti macchine.

In tutti questi casi si può utilizzare un file XML chiamato *server.xml* che consente di configurare il server in modo molto semplice con un metodo dichiarativo. Mediante opportuni tags, infatti, è possibile ad esempio, settare porte diverse da quella di default 8080, oppure configurare alcuni parametri particolari, come ad esempio il tipo di mappaggio tra roles e users per il security domain.

Una volta installato in una opportuna directory che chiamiamo per semplicità TOMCAT_HOME si possono osservare le seguenti sottodirectory che assumono ruoli importanti per la configurazione e l'utilizzo di Tomcat:

- /bin : contiene i file eseguibili di sistema che servono per far partire e per far terminare il server;
- /conf : contiene i file di configurazione del server, compreso il file server.xml di cui si è accennato;
- /lib : contiene tutte le librerie necessarie al funzionamento del server;
- /logs : contiene i file di log di sistema;
- /webapps : contiene tutte le applicazioni Web di cui viene fatto il deployment; queste devono essere organizzate con la struttura gerarchica prevista dalle specifiche Servlet 2.2 e possono eventualmente essere impacchettate sottoforma di file WAR;
- /work : è una directory in cui temporaneamente vengono riportati alcuni file ad uso delle applicazioni, come ad esempio le pagine JSP compilate.

3.1.2 Avvio del server

Una volta installato e configurato, Tomcat deve essere avviato mediante i file eseguibili presenti nella directory TOMCAT_HOME/bin.

Lanciando il file startup.sh (o l'equivalente startup.bat per piattaforme Windows) il server viene avviato. A questo punto è possibile avviare un browser e puntarlo sull'URL di esempi http://localhost:8080/examples per testare la corretta insatallazione di Tomcat. Se tutto è andato a buon fine dovrebbe comparire l'applicazione Web di esempio che viene fornita assieme al server.

3.2 L'EJB container JBoss

JBoss è l'implementazione di un EJB container scritto completamente in Java e disponibile open source on line su SourceForge, un sito che distribuisce una numerosissima quantità di progetti open source e freeware. Per poter utilizzare JBoss come server J2EE è necessario integrarlo con un Web container. Per questo motivo esso viene distribuito sia in versione stand alone, sia assieme a Tomcat o a Jetty, in modo da fornire una full-compilant J2EE platform.

Qualunque sia il tipo di pacchetto scelto (stand alone o assieme ad un Web container) la struttura gerarchica di JBoss è sempre la stessa; supponendo di chiamare JBOSS_HOME la sua directory di installazione, a partire da essa si possono trovare le seguenti sottodirectory:

- /bin : contiene tutti i file eseguibili (sia script che JAR) inclusi nella distribuzione di JBoss per avviare il server;
- /lib e /lib/ext : contengono le librerie Java in formato JAR che JBoss utilizza; quelle in /lib devono essere specificate nel CLASSPATH mentre quelle in /lib/ext sono rese automaticamente disponibili attraverso un classloader (il tipico utilizzo di quest'ultima directory è quello di inserivi i file JAR del driver JDBC utilizzato);
- /db : contiene altre directory con files relativi al database Hypersonic e Instantdb forniti assieme a JBoss per la gestione degli entity beans;
- /deploy : è la directory per l' "hot deployment"; è sufficiente inserire qui i file JAR ed EAR dei moduli e delle applicazioni utilizzati, per ottenerne il deployment automatico;
- /log : contiene i vari file di log del server;
- /conf : contiene i set di files di configurazione; JBoss stand alone viene distribuito con un unico set di files di configurazione posti nella directory /conf/default. In questo modo è possibile aggiungere nuovi set di configurazione personalizzati senza dover modificare quelli di default inserendoli in opportune sottodirectory di

/conf, specificando poi al momento dell'avvio del server quale di questi deve essere utilizzato. La distribuzione di JBoss + Tomcat 4.0 prevede anche un altro set di files di configurazione collocati nella directory /conf/catalina .

- /client : contiene le librerie necessarie ai client stand alone;
- /docs : contiene la documentazione delle API utilizzate da JBoss, e altri documenti in formato HTML;
- /external : contiene altre librerie di utilità;
- /src : contiene il full tree dei sorgenti Java che implementano JBoss;
- /tmp : è una directory di lavoro che contiene file temporanei;

3.2.1 File di configurazione di JBoss

La directory /conf/default contiene il set di file di configurazione con i quali viene distribuita la versione stand alone di JBoss.

Per avviare JBoss con un altro set di configurazioni è sufficiente lanciare il file eseguibile di avvio passando come parametro la sottodirectory di /conf che contiene i files.

Alcuni di questi file utilizzano la tecnologia *JMX (Java Management Extension)*, un set di specifiche e di tool di sviluppo per la gestione in modo standard di ambienti Java, sviluppata dalla Sun Microsystems in accordo con il Java Community Process. Un *Manageble Bean (MBean)* è un oggetto Java che rappresenta una risorsa gestibile attraverso JMX, e segue il modello dei JavaBeans. In particolare i file di configurazione jboss.conf e jboss.jcml consentono di configurare le varie risorse di JBoss attraverso lo standard dei JMX MBean.

Ci sono diversi file di configurazione per JBoss:

- *jboss.properties* : è un file di configurazione nel formato delle Java Properties che viene caricato al momento dello sturtup del server; possono essere specificate in

questo file tutte le proprietà che non necessariamente devono essere disponibili prima dell'invocazione del metodo main di JBoss;

- *jboss.conf* : contiene solo configurazioni di servizi che sono necessari per ottenere il bootstrap iniziale di JBoss come ad esempio l'estensione del CLASSPATH. Questo file è caricato da una istanza della classe `javax.management.loading.MLet` ed è scritto seguendo gli standard di sintassi MLet per i JMX MBeans. Il seguente frammento di file jboss.conf effettua l'estensione del CLASSPATH in modo da includere la libreria jboss.jar contenuta nel percorso /JBOSS_HOME/lib/ext :

```
<MLET CODE="org.jboss.util.ClassPathExtension"
        ARCHIVE="jboss.jar"
        CODEBASE="../../lib/ext/">
   <ARG TYPE="java.lang.String" VALUE="../../log/">
</MLET>
```

- *jboss.jcml* : contiene la lista dei JMX MBeans che devono essere inclusi nella corrente istanza di JBoss; da notare che, contrariamente alla sintassi MLet, questo file è in formato XML. In questo modo si consente la configurazione anche di parametri con un nome, invece che semplicemente coppie TIPO-VALORE ammesse dalla sintassi MLet. Il seguente frammento di file jboss.jcml ad esempio è utilizzato per includere un MailService in JBoss:

```
<mbean code="org.jboss.mail.MailService"
                name="DefaultDomain:service=Mail">
   <attribute name="JNDIName">Mail</attribute>
   <attribute name="ConfigurationFile">mail.properties
   </attribute>
   <attribute name="User">user_id</attribute>
   <attribute name="Password">password</attribute>
 </mbean>
```

In particolare l'attributo `code` dell'elemento MBean indica il nome della classe che deve essere caricata ed istanziata mentre l'attributo `name` dell'elemento MBean specifica un nome unico assegnato al MBean all'interno del dominio JMX. Le dipendenze reciproche degli MBean sono implicitamente evidenziate dall'ordine

in cui appaiono nel file. Perciò ad esempio un MBean di configurazione di un servizio di connection pooling, avendo necessità di far uso di un servizio di naming JNDI, dovrà necessariamente comparire dopo l'MBean di configurazione dei servizi JNDI.

- *jboss-auto.jcml* : in questo file vengono salvate run-time in formato XML tutte le configurazioni degli MBean in uso in una particolare istanza di JBoss. In questo modo se l'amministratore cambia la configurazione dei MBean run-time, nel successivo avvio di JBoss queste modifiche saranno automaticamente caricate.
- *mail.properties* : contiene le proprietà del mail provider (ad esempio dove trovare i server SMTP e POP) che JBoss utilizza in modo da conformarsi agli standard delle specifiche EJB che prevedono che un EJB container garantisca un servizio di mailing attraverso le API JavaMail.
- *jnp.properties* e *jndi.properties* : sono file relativi al servizio di naming JNDI. Il primo contiene la configurazione di JNP, il naming provider utilizzato da JBoss; il secondo contiene le proprietà per i JNDI clients. I clients che intendono far uso del servizio di naming per riferire ad esempio degli EJB, devono contenere questo file in una directory specificata nel local CLASSPATH.
- *standardjaws.xml* : rappresenta un file di configurazione di default per l'engine di CMP (Container Managed Persistance).
- *auth.conf* : questo file contiene indicazioni e configurazioni sui tipi di login modules che JBoss utilizzerà nella gestione della sicurezza. Ad esempio è possibile inserire in questo file un modulo per il mappaggio di roles e users utilizzando tabelle di un database.
- *server.policy* : contiene la configurazione della security policy di JBoss. Di default non viene settata alcuna restrizione, per cui è ammesso ogni tipo di operazione da parte degli EJB. Si può utilizzare questo file ad esempio per inibire accessi a cere zone riservate del FileSystem.

- *standardjboss.xml* : contiene le configurazioni di default del container, come ad esempio il numero di porta che viene riservata alla gestione delle chiamate remote attraverso l'RMI-IIOP che di default è la 4444.

3.3 Configurazione di JBoss e Tomcat nella stessa VM

Finora si è preso in considerazione lo scenario in cui Tomcat e JBoss debbano essere configurati in modo indipendente.

Se invece sia il Web container che l'EJB container vengono installati sullo stesso server J2EE, è molto più conveniente configurarli in modo che possano collaborare venendo lanciati nella stessa VM.

Il pacchetto contenente sia JBoss 2.4.3 chee Tomcat 4.0 , viene distribuito con già un set di configurazione che realizza questo scenario. Nella directory /conf/catalina sono presenti tutti i files di configurazione di JBoss di cui si è accennato nel paragrafo precedente, opportunamente modificati per prevedere l'utilizzo anche del Web container Tomcat. In questo modo, tutte le configurazioni anche del Web container avvengono mediante i file di configurazione di JBoss, per cui il file server.xml di Tomcat viene ignorato.

Si riportano ora le principali caratteristiche di questi file che consentono la collaborazione dei due containers.

- Nel file jboss.conf l'estensione del CLASSPATH viene effettuata in modo da comprendere anche le librerie di Tomcat:

```
<MLET CODE = "org.jboss.util.ClassPathExtension"
   ARCHIVE="jboss.jar" CODEBASE="../../lib/ext/">
  <ARG TYPE="java.lang.String"
       VALUE="../../../catalina/common/lib/">
  <ARG TYPE="java.lang.String"
VALUE="CatalinaCommon">
</MLET>
```

```
<MLET CODE = "org.jboss.util.ClassPathExtension"
   ARCHIVE="jboss.jar" CODEBASE="../../lib/ext/">
  <ARG TYPE="java.lang.String"
       VALUE="../../../catalina/server/lib/">
  <ARG TYPE="java.lang.String" VALUE="CatalinaServer">
</MLET>

<MLET CODE = "org.jboss.util.ClassPathExtension"
   ARCHIVE="jboss.jar" CODEBASE="../../lib/ext/">
  <ARG TYPE="java.lang.String"
VALUE="../../../catalina/bin/">
  <ARG TYPE="java.lang.String" VALUE="CatalinaBin">
</MLET>

<MLET CODE = "org.jboss.util.ClassPathExtension"
   ARCHIVE="jboss.jar" CODEBASE="../../lib/ext/">
  <ARG TYPE="java.lang.String"
VALUE="../../../catalina/lib/">
   <ARG TYPE="java.lang.String" VALUE="CatalinaLib">
</MLET>
```

2) Nel file jboss.jcml la presenza del MBean che configura una classe particolare, la `org.jboss.web.catalina.EmbeddedCatalinaServiceSX`, gestisce i servizi embedded di Tomcat in JBoss, e si occupa di far partire il Web container assieme a JBoss:

```
<mbean
code="org.jboss.web.catalina.EmbeddedCatalinaServiceSX"
       name="DefaultDomain:service=EmbeddedTomcat"
/>
```

La porta di default in cui viene avviato Tomcat è, come sempre, la 8080 ma questa volta per cambiarla occorre agire sull'attributo "Port" di questo MBean e non più sul file server.xml di Tomcat che viene ignorato:

```
<mbean
code="org.jboss.web.catalina.EmbeddedCatalinaServiceSX"
       name="DefaultDomain:service=EmbeddedTomcat">
    <attribute name="Port">8063</attribute>
</mbean>
```

3) Sempre nel file jboss.jcml occorre configurare il MBean che si occupa del deployment in modo che i file WAR vengano opportunamente gestiti dal MBean precedente:

```
<mbean code="org.jboss.deployment.J2eeDeployer"
       name="J2EE:service=J2eeDeployer">
    <attribute name="DeployerName">Default</attribute>
    <attribute name="JarDeployerName">
       :service=ContainerFactory
    </attribute>
    <attribute name="WarDeployerName">
      :service=EmbeddedTomcat
     </attribute>
</mbean>
```

Per far partire JBoss con questa nuova configurazione che comprende anche Tomcat, si può lanciare il comando

```
run_with_catalina.sh
```

3.4 Il connection pooling in JBoss

Per gestire le connessioni al database in JBoss, occorre prima di tutto settare il MBean che definisce i driver JDBC, che, di default, viene configurato con il driver per Hypersonic. Per l'applicazione realizzata sarà necessario aggiungere anche il driver JDBC per SQLServer separato da una virgola dall'altro.

```
<mbean code="org.jboss.jdbc.JdbcProvider"
       name="DefaultDomain:service=JdbcProvider">
   <attribute name="Drivers">
     org.hsqldb.jdbcDriver,
     com.microsoft.jdbc.sqlserver.SQLServerDriver
   </attribute>
</mbean>
```

Per implementare l'interfaccia `DataSource` JBoss utilizza una classe denominata `XADataSourceImpl`, che deve essere opportunamente configurata attraverso un opportuno MBean:

```
<mbean code="org.jboss.jdbc.XADataSourceLoader"

name="DefaultDomain:service=XADataSource,name=SQLServer
Pool">
    <attribute name="PoolName">SQLServerPool</attribute>
    <attribute name="DataSourceClass">
     org.jboss.pool.jdbc.xa.wrapper.XADataSourceImpl
    </attribute>
    <attribute name="Properties"></attribute>
    <attribute name="URL">
      jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433
    </attribute>
    <attribute name="GCMinIdleTime">1200000</attribute>
    <attribute name="JDBCUser">cervellati</attribute>
    <attribute name="MaxSize">10</attribute>
    <attribute name="Password">*******</attribute>
    <attribute name="GCEnabled">false</attribute>
    <attribute name="InvalidateOnError">false</attribute>
    <attribute name="TimestampUsed">false</attribute>
    <attribute name="Blocking">true</attribute>
    <attribute name="GCInterval">120000</attribute>
    <attribute name="IdleTimeout">1800000</attribute>
    <attribute name="IdleTimeoutEnabled">false</attribute>
    <attribute name="LoggingEnabled">false</attribute>
    <attribute
name="MaxIdleTimeoutPercent">1.0</attribute>
    <attribute name="MinSize">0</attribute>
</mbean>
```

In particolare sono stati settati i seguenti attributi:

- URL : il suo valore deve essere la stringa di connessione JDBC che consente al driver di connettersi al server EIS. Nel caso specifico il server EIS è NOUMENO e la porta è quella di default di SQLServer, cioè la 1433.
- JDBCUser : il suo valore de essere il nome utente di accesso al database;

- Password : il suo valore deve essere la password di autenticazione dell'utente;

Per gli altri attributi, tra i quali MaxSize (il numero massimo di connessioni aperte contemporaneamente) e MinSize (il numero minimo di connessioni da aprire all'avvio del server), è stato lasciato il valore di default.

Inoltre sono state aggiunte le librerie di file JAR del driver JDBC nella directory JBOSS_HOME/lib/ext e sono stati compresi i loro nomi nel file di configurazione jboss.conf :

```
<MLET CODE="org.jboss.jdbc.XADataSourceLoader"
     ARCHIVE="jboss.jar,msbase.jar,mssqlserver.jar,msutil
     .jar" CODEBASE="../lib/ext/">
   <ARG TYPE="java.lang.String" VALUE="SQLServerPool">
   <ARG TYPE="java.lang.String"
        VALUE="org.jboss.minerva.xa.XADataSourceImpl">
</MLET>
```

Infine è stato necessario modificare il file di configurazione standardjaws.xml per comprendere la nuova implementazione del pool, ed indicare il nome JNDI da utilizzare nel codice dei componenti:

```
<datasource>java:/SQLServerPool</datasource>
<type-mapping>MS SQLSERVER2000</type-mapping>
```

3.5 La sicurezza su JBoss

Per completare questa rapida carellata sulla configurazione di JBoss, viene accennato ora il criterio di implementazione della sicurezza, che viene detto *JBossSX (Security Extension Framework)*.

Questo sistema è basato su *JAAS (Java Authentication and Authorization Service)* come richiesto dalle specifiche J2EE, e fornisce supporto sia per la gestione dichiarativa che programmatica della sicurezza. In particolare JBoss fa uso di alcune interfacce che forniscono metodi per autenticare un client e mappare gli users ai roles. Implementando queste interfacce si può integrare qualsiasi infrastruttura di gestione della sicurezza. JBoss include una implementazione di default di queste

interfacce, la classe `org.jboss.security.plugins.JaasSecurityManager`, che può ovviamente essere sostituita da una implementazione personalizzata semplicemente cambiando il settaggio del corrispondente MBean :

```
<mbean
code="org.jboss.security.plugins.JaasSecurityManagerService
" name="Security:name=JaasSecurityManager">

    <attribute name="SecurityManagerClassName">
     org.jboss.security.plugins.JaasSecurityManager
    </attribute>

</mbean>
```

Per mappare gli users ai roles esistono diverse possibilità, ciascuna delle quali fa uso di particolari login modules, ossia classi utilizzate per implementare i corrispondenti comportamenti del JaasSecurityManager.

Per ragioni di brevità si riportano i due moduli ritenuti più significativi.

UsersRolesLoginModule

Questo login module fa semplicemente uso di due file di testo in formato Java Properties per l'autenticazione e il role mapping:

- il file users.properties che contiene l'associazione tra i possibili username e le rispettive password;

  ```
  username1=password1
  username2=password2
  ....
  ```

- il file roles.properties che contiene l'associazione tra gli username e i corrispondenti roles separati da virgole;

  ```
  username1=role1,role2, ...
  username2=role3,role4, ...
  ```

```
username2.RoleGroup1=role5,role6,...
...
```

dove la forma username.XXX indica l'assegnamento dei ruoli ad un particolare gruppo di ruoli di nome XXX del utente con quel username, quindi, nell'esempio, vengono assegnati allo username2 i ruoli role3 e role4 senza associarli ad alcun gruppo, e i ruoli role5 e role6 raggruppati in un gruppo di nome RoleGroup1.

Entrambi questi file devono trovarsi nella sottodirectory della particolare configurazione utilizzata assieme agli altri file di configurazione.

DatabaseServerLoginModule

Questo login module fa uso di tabelle di un database per l'autenticazione e il role mapping.

E' basato su due tabelle col seguente aspetto:

PRINCIPALS(PrincipalID , Password)

ROLES(PrincipalID , Role , RoleGroup)

FK : PrincipalID *REFERENCES* PRINCIPALS

Per specificare quale di questi login module devono essere utilizzati dal JaasSecurityManager, occorre far uso di alcuni file di configurazione e di nomi logici del JNDI namespace:

1) nel file auth.conf occorre specificare un nome logico per ciascun login module con la seguente sintassi :

```
nome {
    nome_classe_login_module (required | optional | ... )
    [opzioni]
}
```

2) nel file standardjboss.xml o nel file jboss.xml occorre specificare un nome JNDI del tipo "java:/jaas/xyz" , dove xyz è il nome logico del login module assegnato nel file auth.conf che si vuole utilizzare. Se tale assegnamento è effettuato nel file

standardjboss.xml allora questo sarà applicato a tutte le applicazioni J2EE di cui si fa il deployment; se viene fatto nel file jboss.xml inserito assieme al deployment descriptor ejb-jar.xml nella directory META-INF del context di un modulo EJB di una applicazione J2EE , allora questo sarà applicato ai soli EJB di quel modulo.

Un esempio

A titolo esemplificativo si suppone di voler utilizzare un DatabaseServerLoginModule per autenticare gli user che accedono agli EJB di una applicazione.

Nel file auth.conf deve essere assegnato un nome logico al login module, ad esempio il nome “modulojdbc” :

```
modulojdbc {
      org.jboss.security.auth.spi.DatabaseServerLoginModule
required

      dsJndiName="java:/SQLServerPool"
      principalsQuery="select Password from Principals
where
                        PrincipalID=?"
      rolesQuery="select Role, RoleGroup from Roles where
                  PrincipalID=?"
      ;
};
```

Da notare la possibilità di specificare tre diverse opzioni:

1) dsJndiName : specifica il nome JNDI sotto il quale è stato effettuato il binding del DataSource del database dal quale recuperare le tabelle per l’autenticazione;
2) principalsQuery : specifica la stringa SQL del PreparedStatement da utilizzare per recuperare la password conoscendo lo username;

3) rolesQuery : specifica la stringa SQL del PreparedStatement da utilizzare per recuperare i roles a cui appartiene un user conoscendo il suo username;

Queste opzioni assicurano la possibilità di utilizzare questo tipo di login module qualunque sia la struttura particolare del database sottostante, infatti viene data la possibilità di scegliere da quale database recuperare le informazioni che servirano per l'autenticazione, e come recuperarle.

Nel file jboss.xml del modulo EJB deve essere specificato che il login module da utilizzare è quello di cui sopra:

```
<?xml version="1.0"?>
<jboss>
    <security-domain>java:/jaas/modulojdbc</security-
domain>
</jboss>
```

Utilizzando il tag `<security-domain>` si attribuisce lo stesso login module a tutti i tipi di EJB del modulo EJB, ma esiste la possibilità di differenziarlo a seconda che si tratti di stateless session bean, stateful session bean o entity bean.

Infine per concludere la panoramica sulla sicurezza di JBoss occorre osservare che esiste la possibilità di specificare la stessa configurazione del security-domain anche per la parte Web dell'applicazione. Questo non è possibile farlo attraverso il Web deployment descriptor poiche questo ha una struttura standard ben precisa definita dal DTD fornito dalle Servlet specification, e non è possibile farlo attraverso il file server.xml poiché viene ignorato se JBoss e Tomcat vengono avviati nella stessa VM.

JBoss ha dunque adottato lo stesso criterio visto per gli EJB, e cioè prevedere l'aggiunta di un ulteriore file di configurazione da aggiungere alla directory WEB-INF del Web context dell'applicazione. Questo file ha nome jboss-web.xml :

```
<?xml version="1.0"?>
<jboss-web>
```

```
    <security-domain>java:/jaas/modulojdbc</security-
domain>
</jboss-web>
```

Capitolo 4

Progetto e implementazione dell'applicazione

In questo capitolo viene presentata la progettazione e l'implementazione dell'applicazione delle homepage docenti realizzata.

Viene dapprima riportata una breve descrizione in linguaggio naturale dei principali requisiti e caratteristiche dell'applicazione, seguita da una descrizione più formale attraverso diagrammi Use Case previsti all'interno del modello UML [UML].

Successivamente viene presentata l'introduzione al modello MVC con particolare riferimento alle applicazioni Web, alcuni Class Diagram e Sequence Diagram (sempre del modello UML) particolarmente significativi, e un'ultima parte di implementazione su JBoss e Tomcat con la presentazione del codice Java delle classi principali.

4.1 Scenari dell'applicazione

Il primo passo che è stato affrontato nello sviluppo dell'applicazione è stata la definizione degli scenari che tenessero conto delle varie tipologie di client abilitate ad accedere e delle loro interazioni con l'applicazione stessa.

Ci si è concentrati principalmente su client di tipo Web, anche se, grazie alla architettura adottata con un approccio di tipo EJB-Centric, si è comunque tenuto

conto della possibilità, lasciata eventualmente a futuri sviluppi, di accessi anche da parte di client stand-alone.

Tra i vari Web-client considerati sono state individuate tre categorie principali: l'*utente generico*, l'*utente studente* e l'*utente docente*.

Per utente generico si è inteso l'utente che liberamente si connette alla rete Internet e ha accesso alle sole risorse di interesse generale che non siano soggette ad alcun vincolo di sicurezza. La risorsa Web che rientra in questa categoria è la Homepage di un docente. Essa è infatti accessibile a chiunque senza necessità di autenticazione.

L'utente studente è invece dotato di maggiori possibilità di accesso. Ad egli infatti è concesso di visualizzare la pagina di materiale didattico di un insegnamento.

L'utente docente, infine, è l'unico abilitato ad effettuare modifiche sulla propria homepage e sulla pagina di materiale didattico relativo ai propri insegnamenti.

Per queste ultime due categorie si è resa necessaria una autenticazione attraverso l'inserimento di uno username e una password in un form di login. Questi parametri devono poi essere verificati per consentire l'identificazione dell'utente e concedere l'autorizzazione ad accedere alla risorsa protetta.

4.1.1 Struttura e requisiti della homepage di un docente

Lo scopo della homepage di un docente è quello di fornire all'utente generico informazioni generiche sul docente e su ciò che a lui attiene. La struttura di ogni pagina è suddivisibile in tre parti logiche distinte:

1) dati relativi al docente ed eventuale foto;
2) elenco degli insegnamenti tenuti all'interno dei vari corsi della facoltà di Ingegneria;
3) spazio libero per eventuali pubblicazioni e commenti a cura del docente.

Per quello che concerne il primo aspetto, ovvero i dati relativi al docente, si intende, tra gli altri, il ruolo, il giorno e l'orario di ricevimento, l'ubicazione

dell'ufficio, il numero di telefono, fax e indirizzo email, nonché l'URL di una eventuale pagina Web personale esterna all'applicazione.

L'elenco degli insegnamenti è stato previsto con riferimento agli ultimi tre anni accademici, in modo tale da consentire agli studenti la possibilità di accedere anche al materiale didattico e al programma di anni precedenti. Ogni insegnamento, infatti, è affiancato da due links, uno relativo al programma, e l'altro alla parte di materiale didattico che verrà analizzata in seguito.

La parte di spazio libero è stata progettata prevedendo una suddivisione per argomenti che il docente può arbitrariamente definire. Ciascun argomento ha un titolo e, opzionalmente, una o più note aggiuntive. Inoltre, sempre all'interno di un argomento, il docente può fare riferimenti ad altri siti inserendone il link, o a files precedentemente caricati sul server consentendone il download.

L'utente generico ha libero accesso a questa pagina e ai links dello spazio libero. Può quindi scaricare files e passare ai link riportati all'interno di questa zona, senza alcuna necessità di autenticarsi, a meno che, come meglio specificato in seguito, non ne sia stata esplicitamente interdetta la pubblicazione da parte del docente.

L'utente docente può accedere alla sua pagina anche attraverso un differente URL che ne consente la modifica, previa autenticazione con opportuno form di login, di tutte le sue parti, tranne quelle dei suoi dati, il cui aggiornamento deve avvenire tramite la Segreteria. Una volta autenticato, infatti, egli ha la facoltà di modificare la parte di spazio libero della homepage, e le pagine di materiale didattico dei suoi insegnamenti. Si è inoltre prevista la possibilità di permettere al docente di inserire o cambiare la propria foto e di decidere se pubblicare o meno la parte di spazio libero. Quest'ultima opzione è stata realizzata per consentire al docente di inibire all'utente generico e all'utente studente la visualizzazione dello spazio libero, consentendogli di effettuare le eventuali modifiche che ritiene opportune, e, una volta completato l'aggiornamento, ripubblicare la nuova versione. Le modifiche apportabili su questa parte di spazio libero sono l'inserimento, la modifica e la cancellazione di un qualsiasi elemento (file, link o nota) di ciascun argomento cliccando sul

corrispondente pulsante posto a fianco dell'elemento stesso. Si può anche cancellare un intero argomento compreso tutto il suo contenuto cliccando sul pulsante corrispondente posto accanto al titolo.

4.1.2 Struttura e requisiti della pagina del materiale didattico di un insegnamento tenuto dal docente

Per quanto riguarda la parte di materiale didattico si è previsto l'accesso ai soli utenti facenti parte della categoria di studente. All'utente che vuole visionare il materiale didattico di un certo insegnamento in un certo anno accademico verrà chiesta l'autenticazione attreverso un opportuno form di login. Si è prevista inoltre la possibilità da parte del docente di togliere questo vincolo e rendere la pagina accessibile a chiunque. Per questo motivo nel form di login compaiono già inseriti nei rispettivi campi d'immissione, uno username e una password di default. Se il docente vuole rendere accessibile la pagina potrà abilitare questi parametri di autenticazione di default, nel qual caso l'utente generico avrà accesso semplicemente cliccando sul pulsante di login senza dover aggiungere nulla. Qualora invece il docente volesse interdire l'accesso potrà abilitare nuovi parametri di autenticazione da lui stabiliti e modificabili, rendendo così obbligatorio l'inserimento di username e password corretti da parte dell'utente studente, sempre ferma restando la possibilità di ritornare a quelli di default.

Anche questa pagina ha la stessa organizzazione per argomenti della parte di spazio libero. E' dunque prevista la suddivisione in titoli con le eventuali note, files e links. Se il docente decide di modificarla, deve collegarsi all'URL apposito di cui si è già accennato nella descrizione dell'homepage, autenticarsi come utente docente e cliccare sul bottone posto a fianco di ciascun insegnamento nell'elenco di quelli da lui tenuti negli ultimi tre anni accademici. Questa operazione lo porta alla pagina di materiale didattico corrispondente alla quale sono aggiunti i bottoni di modifica, così come avviene per la parte di spazio libero dell'homepage. Anche per ciascuna pagina

di materiale didattico è prevista la possibilità di inibirne la pubblicazione, in modo che non sia accessibile agli utenti studenti.

4.1.3 Use Case Diagram

Si fornisce ora una rappresentazione schematica degli scenari in modo maggiormente formalizzato cosicchè ne derivi una visione più immediata e meno pesante.

Il metodo di modellazione più adatto a questo scopo è quello degli Use Case Diagrams del modello UML. In essi è infatti possibile esprimere le interazioni degli utenti con l'applicazione con una simbologia chiara e leggibile che astrae da tutti i dettagli implementativi sottostanti. In particolare si è fatto uso di segmenti con frecce unidirezionali per indicare una interazione in sola lettura, e di segmenti senza frecce per indicare interazioni in lettura e modifica.

Use Case della homepage

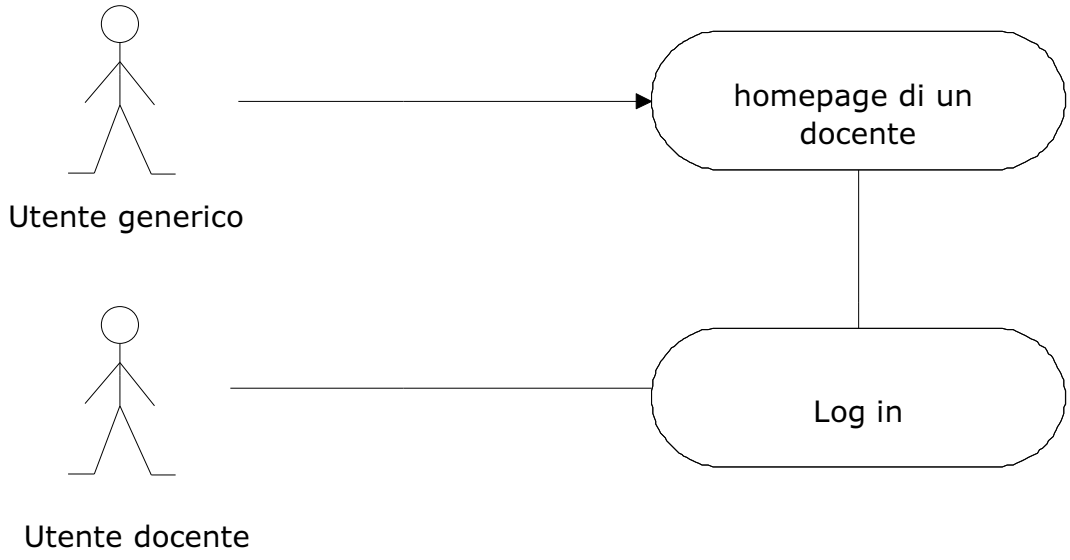


fig 4.1 - Use Case della homepage -

Come si è detto l'utente generico ha libero accesso in sola lettura (freccia monodirezionale) alla homepage di qualunque docente, mentre l'utente docente può accedere in modifica (senza frecce) alla sua homepage previa autenticazione.

Use Case delle operazioni dell'utente generico

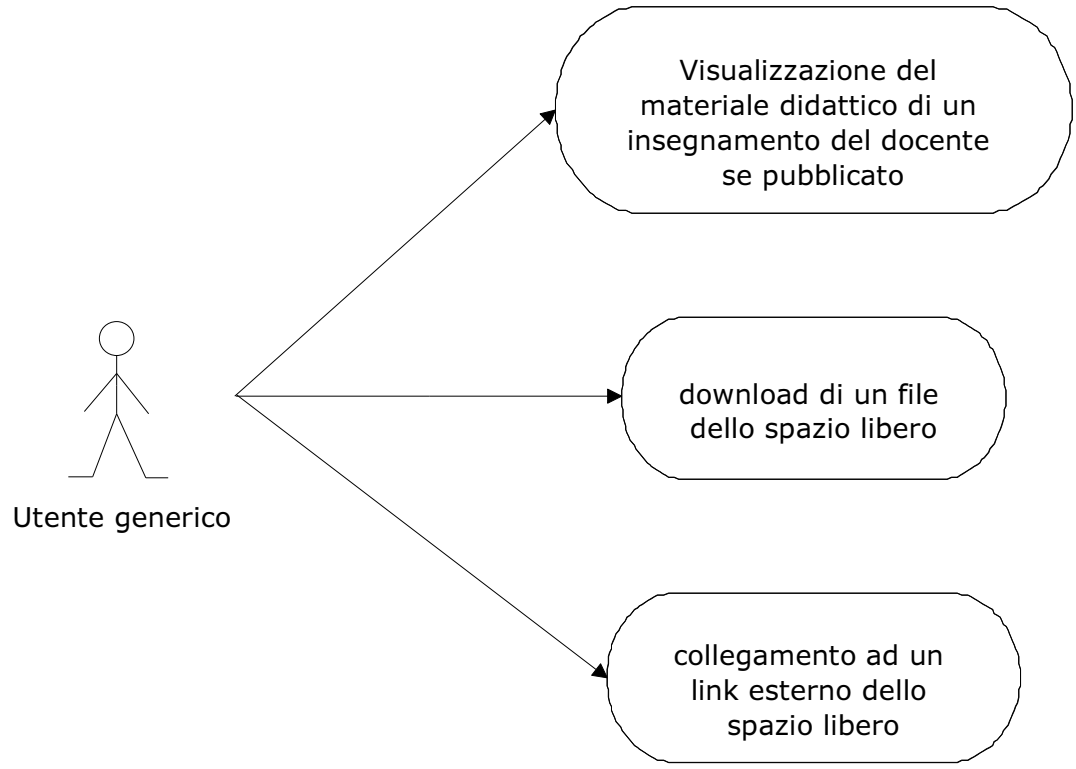


fig 4.2 - Use Case delle operazioni dell'utente generico -

Una volta entrato nella homepage di un particolare docente l'utente generico ha la possibilità di visualizzare la pagina di materiale didattico di un insegnamento tra quelli elencati se il docente ne ha abilitato la pubblicazione, effettuare il download dei files presenti nello spazio libero e collegarsi agli eventuali links esterni presenti sempre nello spazio libero.

Use Case dell'accesso alla pagina di materiale didattico

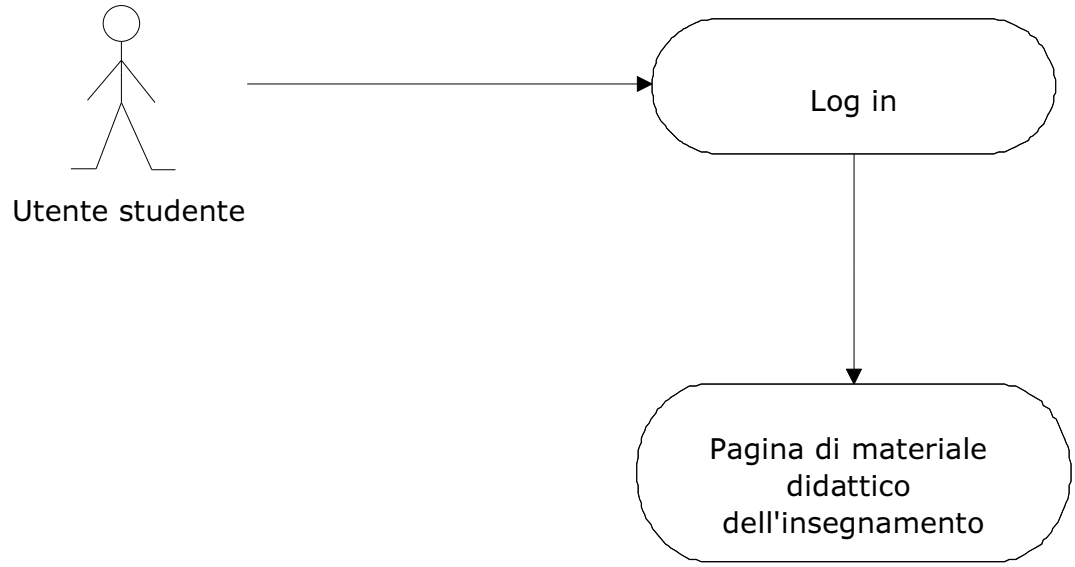


fig 4.3 - Use Case dell'accesso alla pagina di materiale didattico -

Una volta che l'utente generico ha scelto di visualizzare la pagina di materiale didattico di un insegnamento, viene introdotta una procedura di autenticazione attraverso la quale attribuire il ruolo.

Come si è detto il docente può liberamente decidere se consentire l'accesso alla pagina di materiale didattico di ciascun suo insegnamento tramite username e password di default, oppure attraverso username e password da lui stesso definiti. Nel primo caso l'utente generico acquista automaticamente il ruolo di utente studente poiché non necessita di essere a conoscenza di alcun username e password, potendo usare quelle di default già messi a sua disposizione nel form di ingresso.

Use Case delle operazioni dell'utente studente

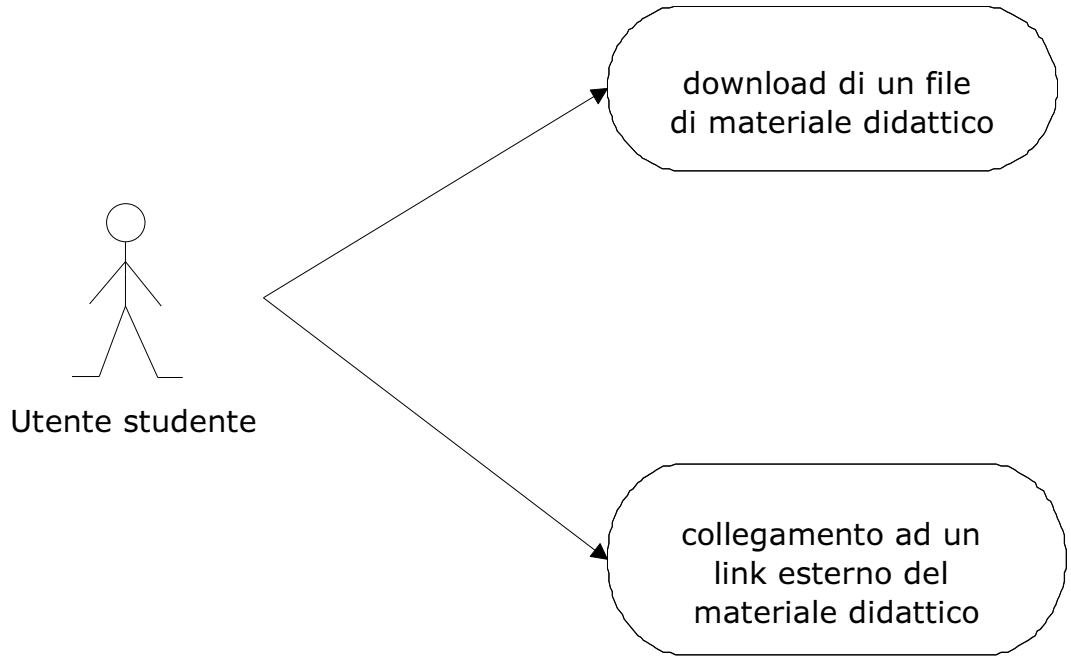


fig 4.4 - Use Case delle operazioni dell'utente studente -

Una volta autenticato l'utente studente può effettuare sulla pagina di materiale didattico le stesse operazioni che poteva effettuare l'utente generico sulla parte di spazio libero dell'homepage.

Use Case delle operazioni del docente

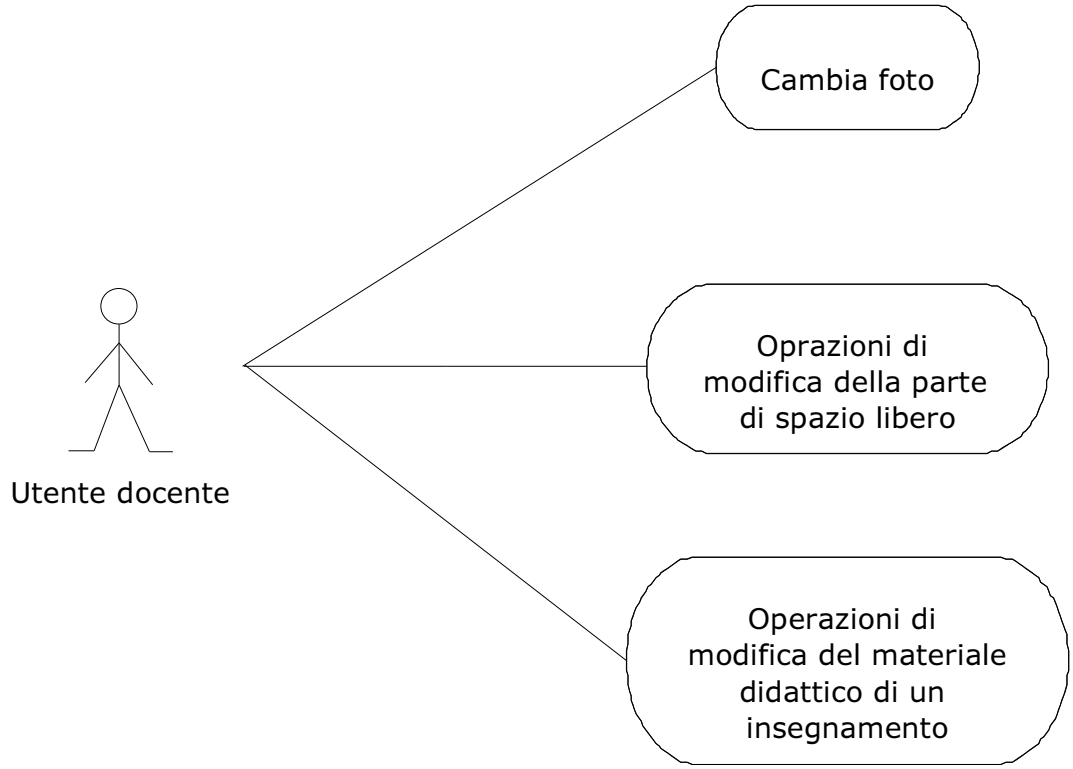


fig 4.5 - Use Case delle operazioni del docente -

Anche l'utente docente, con l'operazione di login, viene automaticamente indirizzato alla propria homepage, nella quale ha la possibilità di effettuare una serie di operazioni di modifica. Principalmente può agire su tre fronti: sulla foto, sulla parte di spazio libero e sul materiale didattico di ciascun insegnamento.

Use Case delle operazioni del docente sulla parte di spazio libero

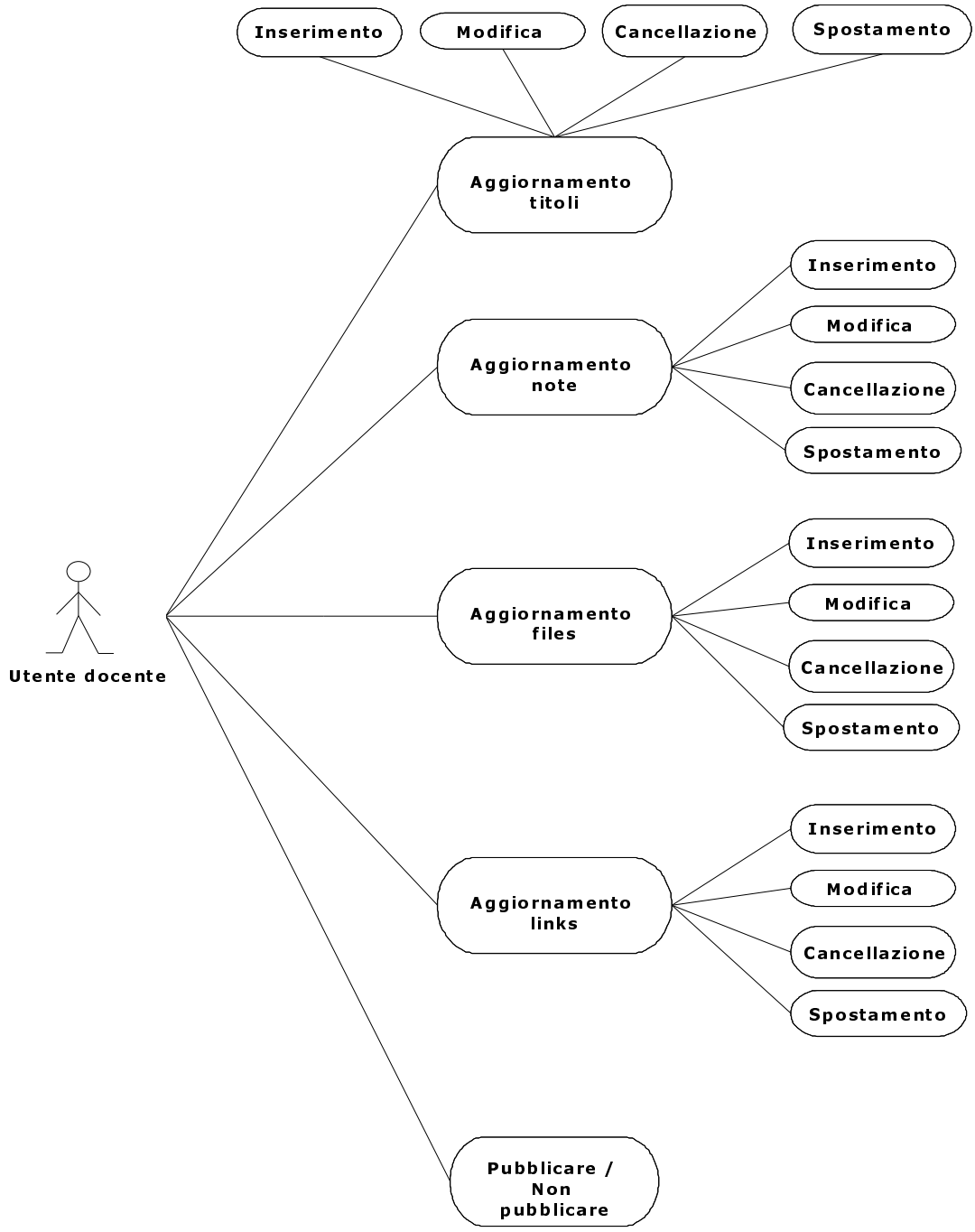


fig 4.6 - Use Case delle operazioni del docente sulla parte di spazio libero -

Le operazioni che il docente può fare sulla parte di spazio libero sono di inserimento, modifica, cancellazione e spostamento di uno o più elementi della struttura. In particolare l'operazione di spostamento consente di mutare a piacimento l'ordine di visualizzazione degli oggetti.

L'altra operazione consentita è quella di rendere pubbliche o meno le modifiche effettuate.

Lo Use Case delle operazioni del docente sul materiale didattico di un insegnamento sono le stesse qui riportate per lo spazio libero, essendo la struttura del tutto analoga, pertanto non viene riportato il diagramma. Va però tenuto presente che per il materiale didattico il docente può anche decidere di modificare lo username e la password di accesso.

4.2 Object decomposition

Una volta definito lo scenario dell'applicazione si è passati alla identificazione concettuale dei diversi oggetti che la compongono, cercando di individuarne il livello di appartenenza.

Nell'effettuare questa suddivisione si è cercato di garantire alcuni dei requisiti fondamentali della progettazione object-oriented, quali la riusabilità del codice e l'identificazione delle responsabilità di ciascun oggetto in modo non ambiguo. Inoltre, nell'ottica di sviluppo di una applicazione distribuita multilivello si è cercato di mantenere sempre in primo piano alcuni obiettivi fondamentali come la separazione tra il codice cosiddetto stabile (cioè soggetto a modifiche poco frequenti) e quello più volatile (cioè soggetto a frequenti modifiche), e la divisione dei contributi delle varie figure professionali coinvolte nel progetto (ad es. gli HTML layout e graphic designer, i programmatori, gli esperti di database, ecc.).

4.2.1 Class Diagram

Un primo approccio alla *object decomposition* avviene con l'ausilio dei Class Diagram dell'UML, che hanno il compito di fornire una prima indicazione di massima sugli oggetti dell'applicazione e le relazioni esistenti tra loro.

Molte delle informazioni utili per questa fase derivano direttamente dallo schema relazionale del progetto.

La prima classe utile, parlando delle homepage dei docenti, è il docente. Ogni docente tiene un certo numero di insegnamenti in un certo anno accademico, e così vengono individuate altre due classi.

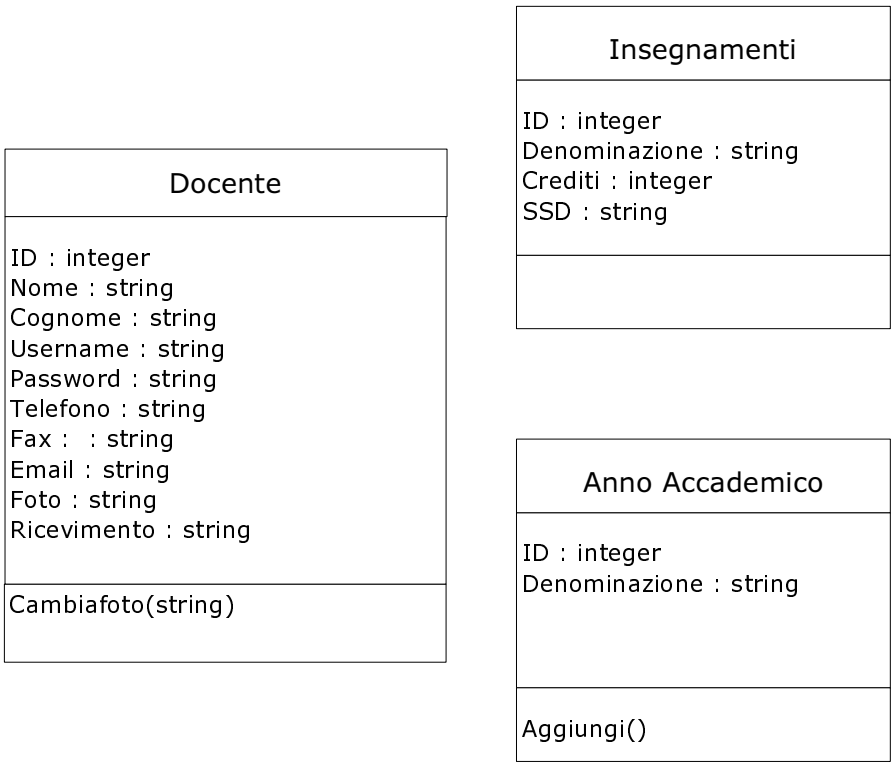


fig 4.7 - Alcune classi fondamentali -

Per vedere come queste si relazionano tra loro introduciamo un'altra classe che rappresenta l'elenco degli insegnamenti in un certo anno accademico:

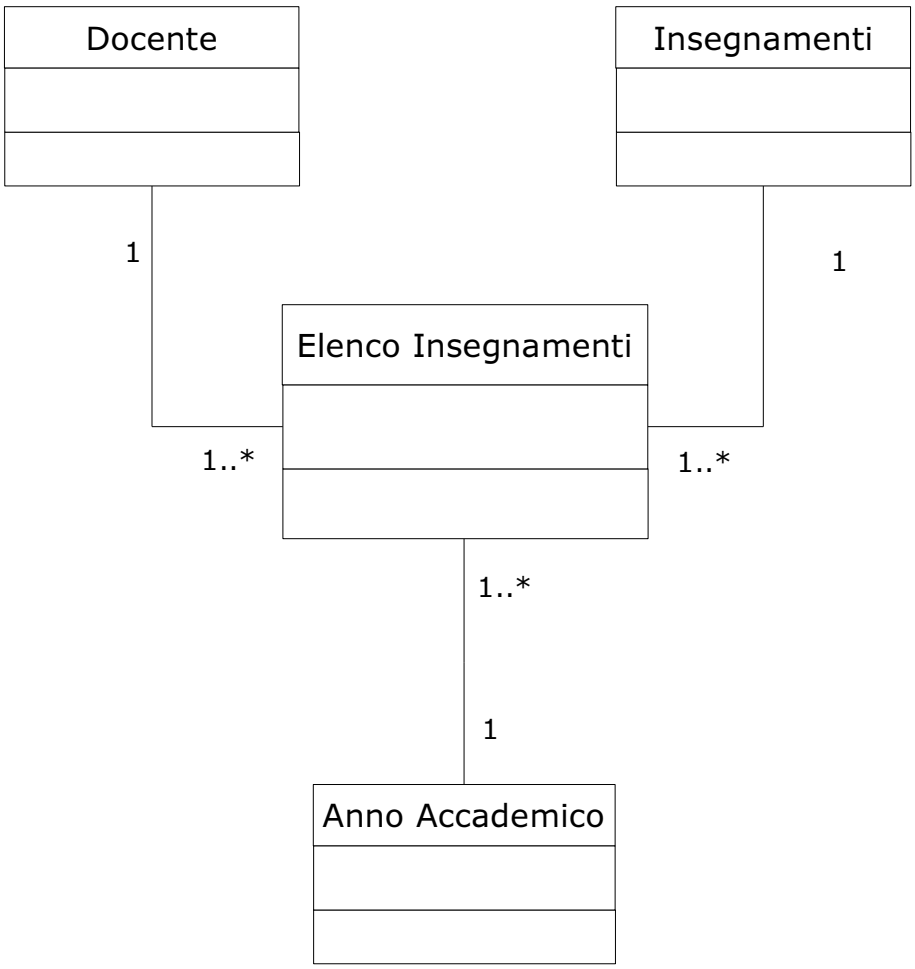


fig 4.8 - Classi principali e loro relazioni -

A questo punto si può entrare più nello specifico della applicazione, osservando che l'elenco degli insegnamenti è una delle parti previste nella homepage di un docente, e rimane da modellare con una opportuna classe la parte di spazio libero, mentre la parte di dati del docente è sostanzialmente già descritta dalla classe docente.

Ogni insegnamento presente nell'elenco può inoltre avere associato del materiale didattico, che viene modellato anch'esso da una nuova classe.

Si osserva che la parte di spazio libero e la parte di materiale didattico sono molto simili poiché sono strutturate allo stesso modo. Si può dunque modellare questo oggetto con una superclasse denominata "Strutturatitoli".

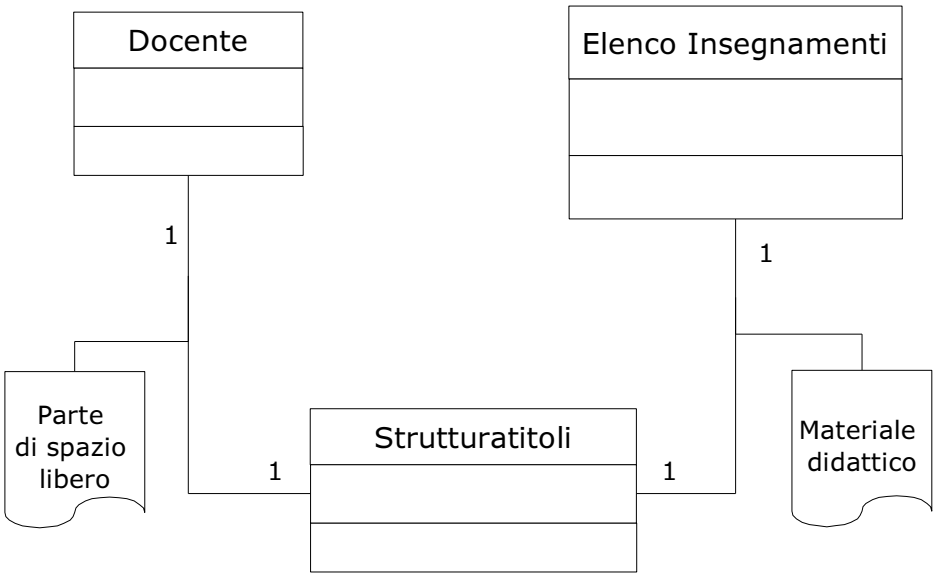


fig 4.9 - La classe "Strutturatitoli" -

La classe Strutturatitoli può essere raffinata tenendo presente che è composta da titoli, e ciascuno di questi da files, note e links.

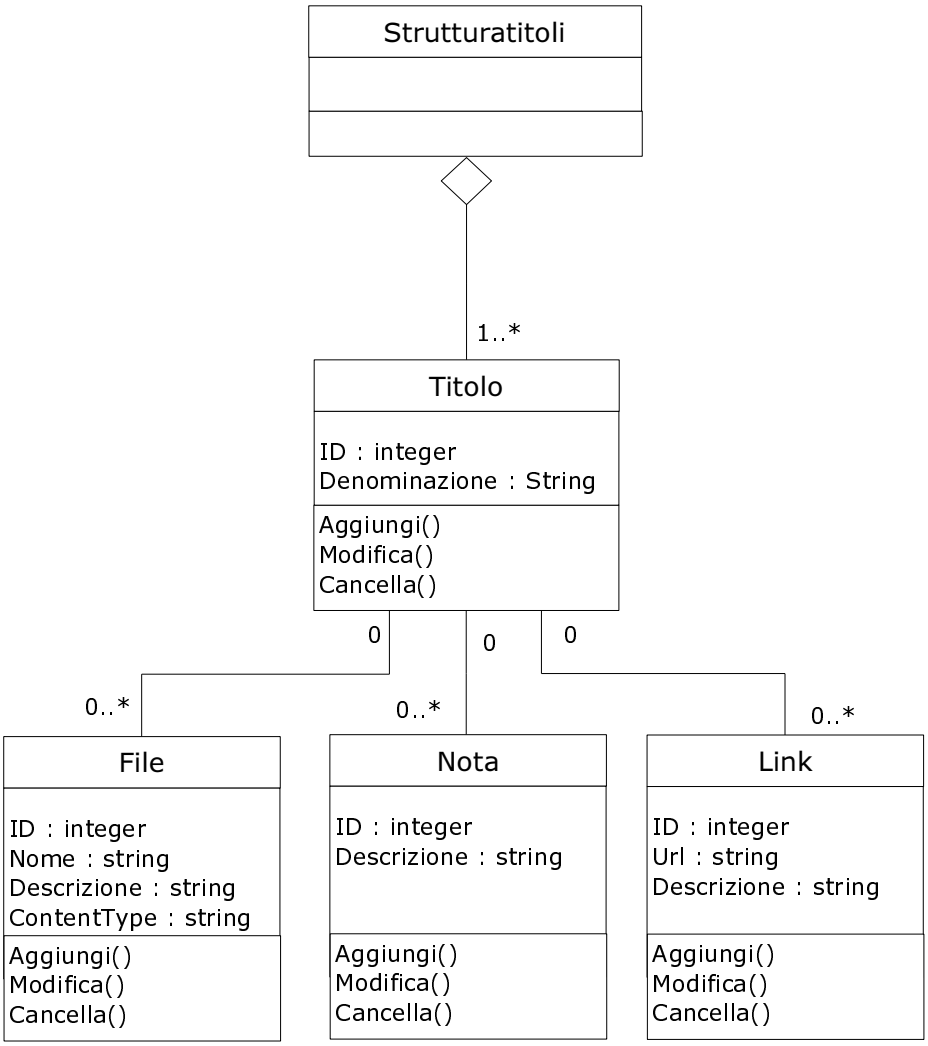


fig 4.10 - Raffinamento della classe "Strutturatitoli" -

4.2.2 Sequence Diagrams

Finora si è data una visione statica dell'applicazione, in termini di oggetti e funzioni. Per avere una visione dinamica, cioè una visione legata a sequenze temporali di azioni, l'UML viene in aiuto con i Sequence Diagrams.

Può essere interessante esaminare ad esempio la sequenza temporale di azioni che coinvolgono una operazione di download di un file di materiale didattico da parte di un utente studente.

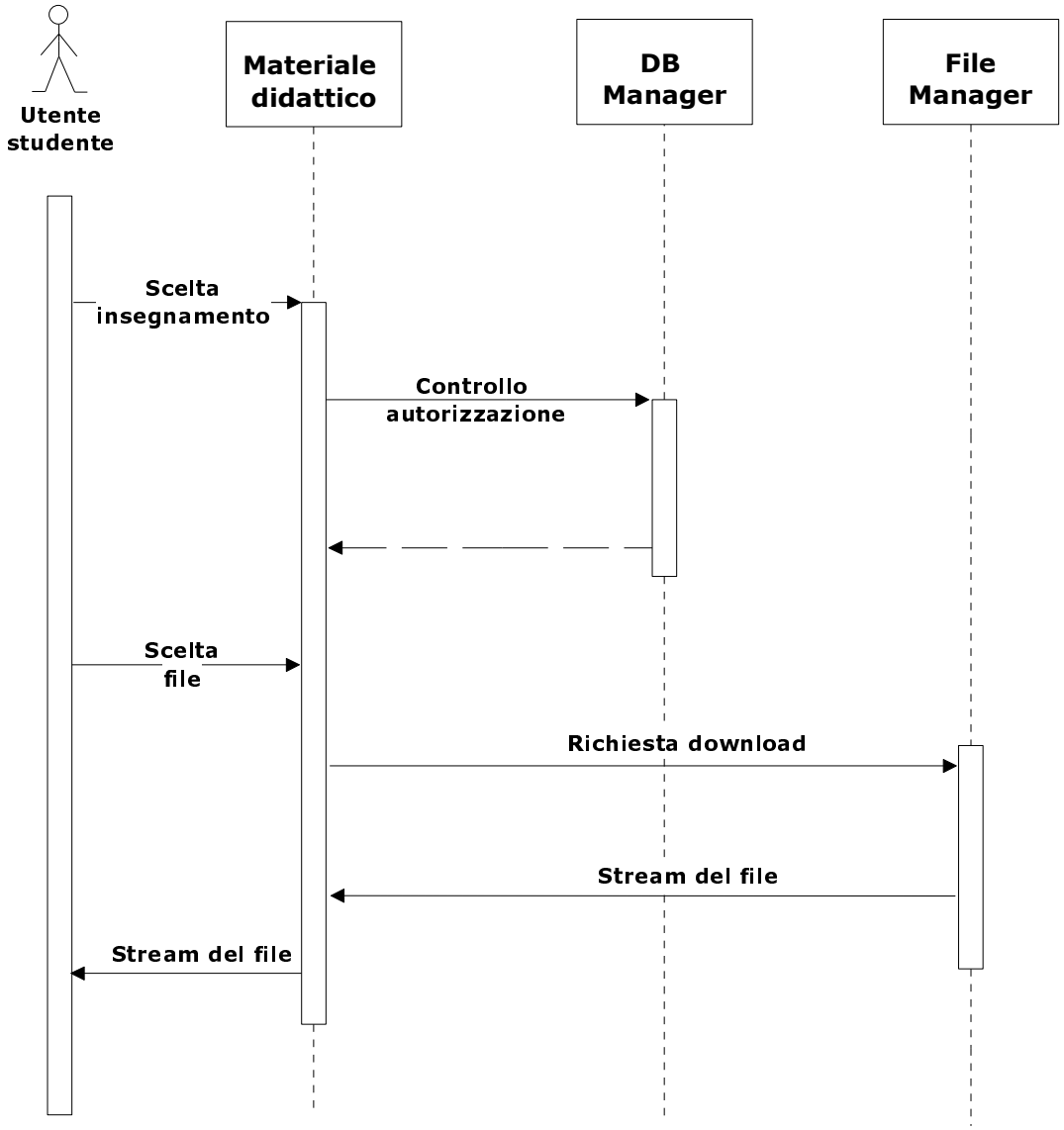
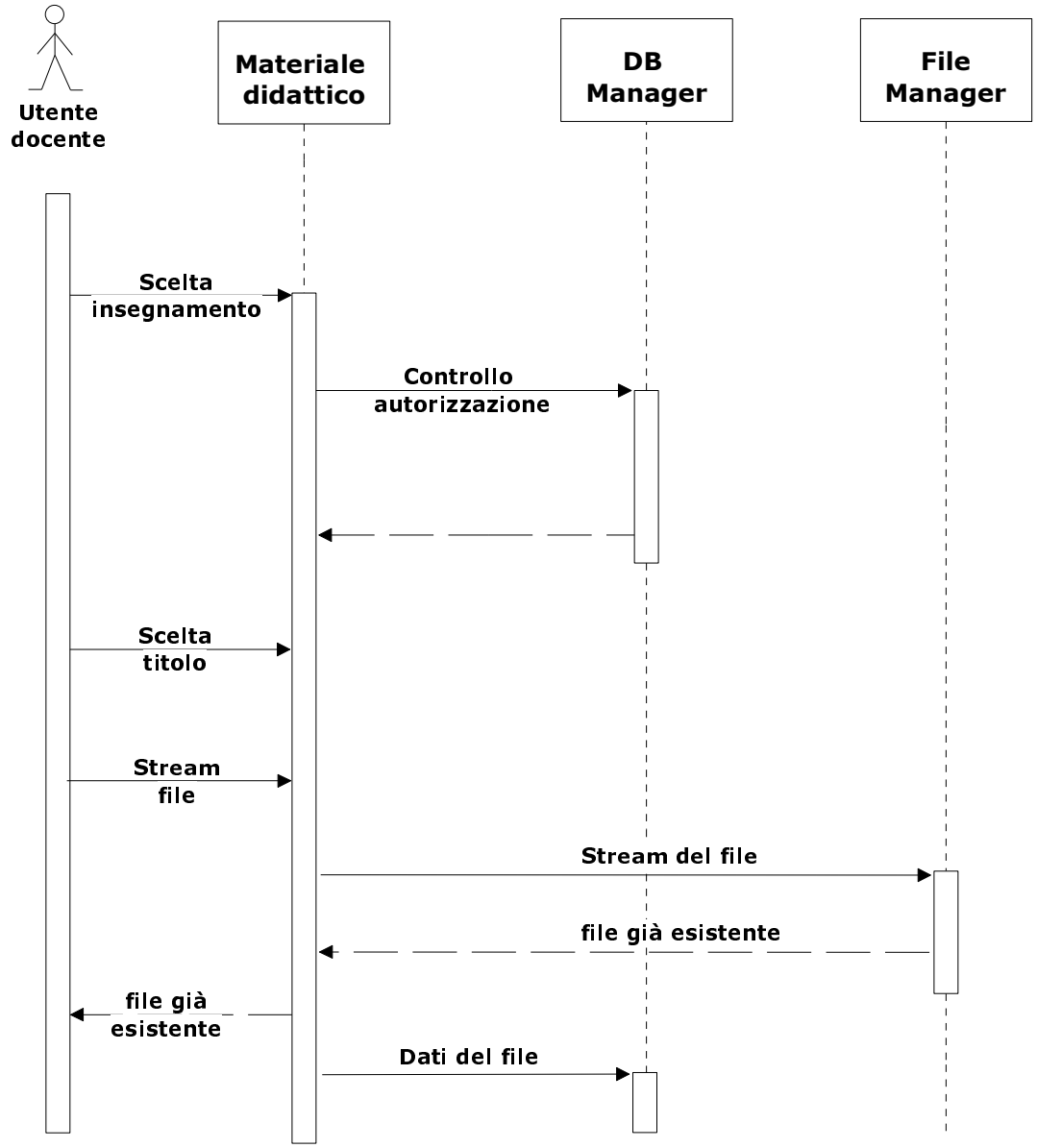


fig 4.11 - Sequence Diagram dell'operazione di download di un file -

A titolo esemplificativo si riporta anche il Sequence Diagram di una operazione effettuata dal docente, in particolare l'upload di un file nel materiale didattico di un suo insegnamento.

Si osservi la eventualità del verificarsi di una condizione in cui il file che il docente desidera caricare abbia già un omonimo presente nel File System. E' questo il caso in cui all'utente deve esserne data notizia e attesa conferma.



4.12 - Sequence Diagram dell'operazione di upload di un file -

4.2.2 Architettura MVC

L'architettura *Model-View-Controller (MVC)* [MVC] costituisce un valido criterio di design di una applicazione J2EE per garantire la separazione tra

funzionalità di business logic e accesso ai dati, dalla presentation logic. In particolare, risulta essere ideale per collocare gli oggetti individuati nella prima fase di object decomposition nei tre livelli logici di cui si compone, e consente ulteriori raffinamenti per arrivare a livelli sempre più vicini all'implementazione vera e propria:

1) **Model** rappresenta i dati sui quali una applicazione si appoggia. Possiamo in linea generica identificare i model-objects con i business-objects di cui si è parlato a proposito della business logic. In un approccio EJB-Centric gli Enterprise Java Beans modellano questi oggetti e ne gestiscono gli aggiornamenti.
2) **View** fornisce una rappresentazione dei dati contenuti nel Model in uno specifico formato. La View di una applicazione Web di tipo enterprise è, come si è detto, costituita principalmente da JSP.

   In particolare i dati modellati nel Model da ciascun EJB sono riflessi in un corrispondente JavaBean che ne fornisce la rappresentazione nel View consentendo alla JSP nella quale sono inseriti di visulalizzarne il contenuto in modo semplice. E' importante sottolineare che tali JavaBean hanno compiti puramente di presentazione dei corrispondenti EJB, mentre a quest'ultimi spetta il compito di aggiornare i dati da essi incapsulati e gestire la business logic dell'applicazione.
3) **Controller** fa da tramite tra il Model e il View, nel senso che si occupa di far sì che l'immagine presentata dal View sia coerente con i dati contenuti nel Model e si occupa di convertire le interazioni che l'utente fa sulla View in corrispondenti eventi di aggiornamento sul Model.

   Dal momento che il Controller ha questo ruolo di coordinatore tra il Model e il View, esso è solitamente costituito da componenti che rientrano sia nel Web-tier che nel EJB-tier.

4.3 Implementazione

Si è giunti all'ultima fase dello sviluppo di questa tesi, cioè all'implementazione dell'applicazione facendo uso della J2EE platform fornita dall'accoppiata JBoss 2.4.3 - Tomcat 4.0 , in un ottica component-based basata sull'architettura MVC.

4.3.1 Model

Gli oggetti Model sono legati al back-end dell'applicazione, che può essere visto come una collezione di "stati" con una serie di regole secondo le quali questi possono variare in risposta alle interazioni dell'utente.

Come già detto queste insiemi di stati e regole costituiscono la business logic dell'applicazione e gli strumenti che la J2EE mette a disposizione per implementare questi oggetti sono gli EJB.

Non sempre è opportuno che tutti i business-object del EJB-tier siano implementati attraverso degli EJB. Essendo oggetti remoti, essi infatti consumano una notevole quantità di risorse di sistema e di rete. Esistono allo scopo una serie di oggetti denominati *Helper objects*, che vengono in "aiuto" al progettista per implementare certe tipologie di oggetti evitando di usare gli EJB : i *Data Access Object (DAO)* [DAO] e i *Value Object (VO)* [VO] .

I DAO sono oggetti che incapsulano l'accesso al database, mantenendolo separato dagli altri oggetti dell'EJB-tier. I principali vantaggi derivanti da tale separazione sono i seguenti:

1) mantenere il codice degli altri oggetti più chiaro e semplice, consentendo una più mirata attenzione alla business logic;
2) consentire una più semplice ed agevole portabilità dell'applicazione da un database ad un altro, dovendo modificare solamente i DAO.

I VO possono invece essere utilizzati per implementare dei business-object con struttura immutabile e funzionalità molto semplici, tipicamente una serie di attributi

con dei metodi getter per recuperarli. Essi sono implementati con oggetti Java serializzabili che vengono passati per valore al client.

Per cominciare ad implementare i business-object il punto di partenza è quello di riprendere il Class Diagram, raffinarlo se necessario poiché rappresenta un modello di massima, e stabilire quale degli strumenti qui accennati utilizzare per implementarlo.

L'oggetto Docente contiene tutti i dati relativi ad un docente che devono comparire sulla homepage. Ogni istanza di questo oggetto incapsula i campi di ciascuna tupla della tabella TBL_DOCENTI; grazie alla sua struttura semplice e immutabile, e alle funzionalità limitate, si è deciso di implementare questo oggetto con un VO.

Un altro busines-object individuato è l'Elenco Insegnamenti (o più semplicemente Insegnamenti) che, come richiesto nelle specifiche dei requisiti della homepage, deve riportare gli insegnamenti degli ultimi tre anni accademici tenuti dal docente.

Per quanto riguarda l'oggetto Struttura Titoli, il Class Diagram va raffinato poiché occorre definire due oggetti che ereditano da questo la loro struttura: l'oggetto Spazio Libero e l'oggetto Materiale Didattico. Essi, pur avendo una struttura analoga, presentano una differenza sostanziale, infatti il Materiale Didattico non è relativo ad un docente, ma ad un insegnamento di un certo corso in un certo anno accademico. Esiste infatti una potenziale pagina di materiale didattico per ogni tupla della tabella TBL_INSEGNAMENTI_AA, e il legame con il docente associato è derivato dal join con la tabella TBL_DOCENTI_INSEGNAMENTI_AA. Ad ogni istanza di Docente corrisponde una unica istanza di Spazio Libero della sua homepage, ma corrispondono più istanze di Materiale Didattico, una per ogni insegnamento elencato in Insegnamenti, e quella che deve essere mantenuta nello stato del client e quella che di volta in volta viene da lui richiesta.

Gli oggetti Insegnamenti, Materiale Didattico e Spazio Libero sono stati implementati con degli stateful session beans, di nome InsegnamentiBean,

MatDidBean e SpLiberoBean rispettivamente. La scelta è caduta sugli stateful e non sugli stateless poiché deve essere mantenuto lo stato conversazionale del client attraverso chiamate a più metodi all'interno della stessa sessione. Per ragioni di semplicità si è deciso di far agire direttamente i metodi dei session beans sul database saltando l'ulteriore passaggio, progettualmente più corretto, di incapsulare le varie tabelle del database con degli entity beans.

Una volta individuati i vari business-object dell'applicazione, si è passati alla scelta del criterio per mantenere i dati al loro interno, alla definizione dei metodi per modificarlo e ai criteri di comunicazione con gli oggetti del View.

Per quanto riguarda il VO Docente, questo non ha presentato particolari problematiche essendo semplicemente una collezione di attributi di tipo analogo a quelli delle tuple della corrispondente tabella. Esso è stato implementato come una classe Java serializzabile con soli metodi getter e setter sui vari attributi.

Più complessa è la struttura degli oggetti implementati con gli EJB, e più complesso è anche il metodo per trasferire tutti i dati in essi contenuti ai View-object per essere presentati al client. La scelta ottimale per trasferire in blocco una considerevole quantità di dati semistrutturati è quella di utilizzare il linguaggio XML. Tutti e tre gli EJB sono dunque stati implementati in modo da mantenere le informazioni in una struttura gerarchica ad albero e prevedendo un metodo che trasformasse tale struttura in una stringa XML da inviare all'Web-tier. Java mette a disposizione diverse API per la gestione dei dati in formato XML; la scelta è caduta sulle API *JAXP (Java API for XML Processing)* [JAXP] e in particolare su quelle che fanno uso della struttura *DOM (Document Object Model)* [DOM] che forniscono una serie di classi adatte ad incapsulare i dati in una struttura ad albero nella quale ogni tag rappresenta un nodo e che può essere facilmente convertita in XML.

Grazie a questa struttura comune a tutti gli EJB, si è introdotta una superclasse ContenutoDOM dalla quale ereditano, che contiene una variabile di istanza di tipo `Document` che mantiene la struttura ad albero e due metodi, `Carica()` per

caricare i dati nel `Document` prendendoli dal database, e `getDOM()` che restituisce la stringa XML del contenuto.

Il Class Diagram raffinato su questi ultimi concetti è il seguente:

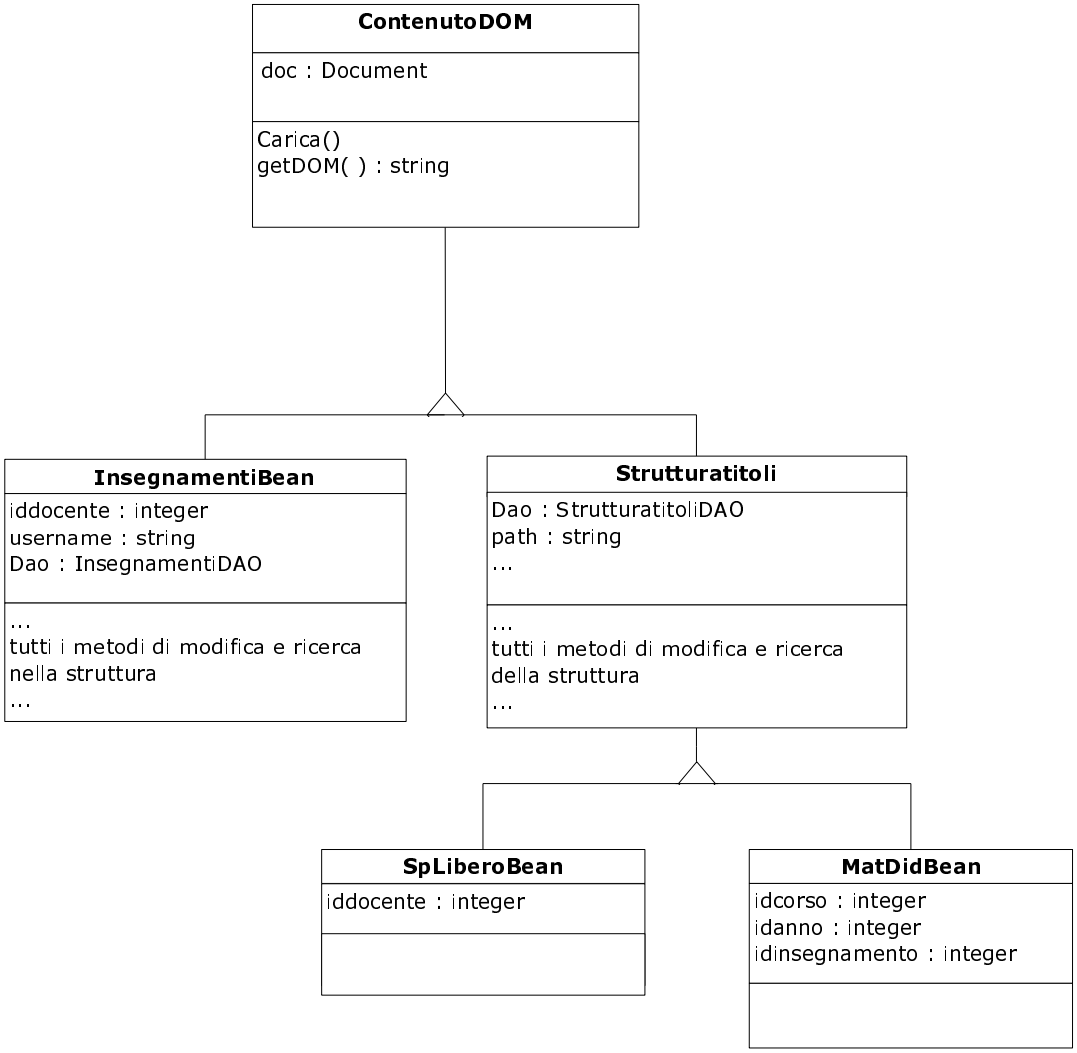


fig 4.13 - Raffinamento delle classi degli EJB -

Per chiarezza vengono riportati qui di seguito due esempi di stringa XML contenente i dati dei bean, il primo dei quali si riferisce al bean InsegnamentoBean della homepage di un docente ipotetico. Essendo la struttura della parte di spazio

libero del tutto analoga a quella del materiale didattico di un insegnamento, viene riportato un esempio di stringa XML che si può ottenere dalla superclasse Strutturatitoli.

```
<insegnamenti>
       <anno-accademico id = "3" den = "2002/03">
          <insegnamento id = "15" programma = ""
pubblicata = "1">
               <denominazione>Fondamenti di Informatica A
               </denominazione>
               <corso id = "1">Corso di Laurea in
               Ingegneria
                              Informatica
               </corso>
          </insegnamento>
          <insegnamento id = "16" programma = ""
pubblicata = "1">
               <denominazione>Fondamenti di Informatica
B
               </denominazione>
               <corso id = "1">Corso di Laurea in
          Ingegneria
                              Informatica
               </corso>
          </insegnamento>
       </anno-accademico>
     <anno-accademico id = "2" den = "2001/02">
          <insegnamento id = "20" programma = ""
pubblicata = "1">
               <denominazione>Basi di Dati
A</denominazione>
               <corso id = "2">Diploma in Ingegneria
                              Informatica
               </corso>
          </insegnamento>
       </anno-accademico>
     <anno-accademico id = "1" den = "2000/01">
          <insegnamento id = "32" programma = ""
pubblicata = "0">
```

```
                <denominazione>Fondamenti di Informatica
C
                </denominazione>
                <corso id = "1">Corso di Laurea in
          Ingegneria
                                Informatica
                </corso>
          </insegnamento>
       </anno-accademico>
</insegnamenti>
```

- Stringa XML di esempio dell'elenco insegnamenti di un docente -

```
<pag pubblicata = "1">
     <titolo id = "1">
          <denominazione>Database
relazionali</denominazione>
          <file id = "1" idf = "1" nome = "DBMS.doc"
               content-type = "application/msword">
                    introduzione ai DBMS (file Word)
          </file>
          <link id = "1" idl = "1" url =
"http://microsoft.com">
               Sql Server della Microsoft
          </link>
          <link id = "2" idl = "2" url = "">
               elenco driver JDBC
          </link>
     </titolo>
     <titolo id = "2">
          <denominazione>Esercitazioni
svolte</denominazione>
          <nota id = "1" idn = "1">
            tutti gli esercizi riportati sono stati compiti
          d'esame
          </nota>
          <file id = "2" idf = "1" nome = "ese1.txt"
               content-type = "text/plain">
```

```
                    Compito del 1/2
            </file>
            <file id = "3" idf = "2" nome = "ese2.txt"
                content-type = "text/plain">
                    Compito del 3/5
            </file>
            <link id = "3" idl = "1"
                url = "http://www.java.sun.com">
                    Sito di Java
            </link>
        </titolo>
</pag>
```

- Stringa XML di esempio di struttura di Strutturatitoli, comune ai due bean SpLiberoBean e MatDidBean -

Un commento particolare merita la struttura XML di Strutturatitoli. Si noti che in tale struttura sono stati riportati sottoforma di elementi body tutte le denominazioni di titoli, files, note e links contenuti nelle rispettive tabelle del database, mentre gli altri campi sono stati riportati come attributi. In particolare i campi IdFile, IdNota e IdLink sono stati introdotti come attributi dei tag cui si riferiscono, e denominati rispettivamente 'idf', 'idn' e 'idl'. Facendo parte di un identificatore esterno assieme al titolo di appartenenza, questi attributi non sono unici all'interno di una stessa pagina di materiale didattico. E' risultato molto comodo introdurre anche degli ulteriori attributi, denominati 'id', che avessero il carattere di unicità per un certo tag all'interno della stessa pagina. Le API DOM forniscono infatti dei metodi per ottenere una lista di tutti i nodi rappresentati da un certo tag presenti come discendenti di un tag parente. Una ricerca su un nodo file, risulta così agevolata grazie alla possibilità di individuare direttamente all'interno della lista di tutti i nodi file nipoti del nodo pag, quello di cui si conosce l'attributo id. Gli oggetti del View potranno fare riferimento direttamente a questi identificatori per indirizzare le richieste dell'utente sull'opportuno elemento. L'IdTitolo essendo già unico

all'interno di una pagina, non ha richiesto l'introduzione di un ulteriore attributo, ed è rimasto indicato con il nome id per unicità di nomenclatura.

Infine, per incapsulare tutti gli statement SQL di accesso alle tabelle del database, si sono introdotti tre DAO: DocenteDAO, InsegnamentiDAO e StrutturaTitoliDAO. Quest'ultimo, dovendo agire su tabelle diverse a seconda della sottoclasse di Strutturatitoli che ne fa uso, prevede due metodi che inizializzano in modo opportuno le variabili contenenti le stringhe SQL da utilizzare, che verranno richiamati nel momento in cui viene istanziato.

Il codice di un metodo di modifica di ogni EJB realizzato è stato organizzato suddividendolo in due parti principali più una terza opzionale:

1) aggiornamento nel database
2) aggiornamento nella struttura DOM
3) eventuale aggiornamento nel file system

Le eventuali modifiche nel file system sono limitate agli EJB che ereditano da Strutturatitoli, essendo prevista la cancellazione e l'inserimento di file da parte dei docenti. Per realizzare l'upload sono state utilizzate delle librerie della "*O'Reilly - books, conferences and online publishing*" [OREILLY] , organizzazione che mette a disposizione software open source in rete per gli sviluppatori web. Le classi Java messe a disposizione da tale libreria sono in grado di gestire in modo molto semplice le request di tipo *multipart/form-data*[6], che contengono oltre a parametri classici, anche files. Proprio per il fatto che il file viene spedito dall'utente docente assieme al parametro che contiene la descrizione, si è attribuito al Web-tier il compito di gestire la request di tipo multipart/form-data, scomporla nelle sue componenti e rispedirle separatamente all'EJB-tier. Si rimanda alla discussione sui View-objects per ulteriori dettagli.

[6] E' questo il tipo di contenuto del form HTML utilizzato per l'upload di files. Vedi appendice B.

4.3.2 View

Gli oggetti View riguardano il front-end dell'applicazione, cioè il modo in cui all'utente vengono presentati i dati del Model. Come già detto in precedenza la J2EE raccomanda l'uso di JSP con la minor quantità possibile di scriptlet, incapsulando il codice Java in componenti quali JavaBeans e Custom tags.

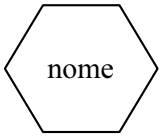
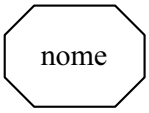
L'applicazione realizzata fa uso di JavaBeans per incapsulare la presentation dei corrispondenti EJB. Ogni JavaBean, infatti contiene un riferimento al suo corrispondente EJB, e si occupa del suo istanziamento e inizializzazione, nonché della sua rimozione quando non più richiesto.

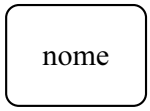
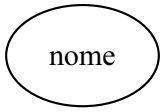
Per la presentazione dei dati provenienti dai Model-objects sottoforma di stringhe XML, si è deciso di fare uso di particolari librerie di tag fornite open source dal progetto Jakarta. Il progetto Jakarta mette, infatti a disposizione una serie di Custom tags organizzati in librerie diverse a seconda della loro funzionalità. Tra queste ne esiste una denominata *XSLTaglib* [XSLTL] , che fornisce dei tag per la trasformazione di risorse XML attraverso dei fogli di stile XSL. Inserendo questo tag in un punto qualunque di una JSP si può ottenere la generazione dinamica di HTML a partire dalla sorgente XML e dal file XSL specificati.

Trattando degli oggetti del View e avendo a che fare con client di tipo Web, è parso opportuno mostrare in questa sede anche le varie interazioni con gli utenti, e si è fatto uso anche di schemi a blocchi riassuntivi dei vari scenari individuati.

La legenda della simbologia adottata in questi schemi è riportata qui di seguito:

SIMBOLOGIA DEI COMPONENTI

SIMBOLO	COMPONENTE
	servlet
	JSP
	JavaBean

SIMBOLO	COMPONENTE
	EJB
	DAO
	risorsa EIS

SIMBOLOGIA DI COMUNICAZIONE

SIMBOLO	COMPONENTE
	passaggio request
	passaggio dati tra componenti

fig 4.14 - Legenda della simbologia per gli schemi a blocchi -

Scenario dell'utente generico

Nello scenario dell'applicazione si è considerato dapprima l'utente generico che visualizza l'homepage di un docente. Questa pagina è realizzata da una JSP di nome homepagedocente.jsp. Essa fa uso di due JavaBean, di nome presentazioneInsegnamenti e paginaSplibero. Prima di accedere alla JSP la request

passa attraverso una servlet di nome hpdoc che riceve come parametro l'id del docente cui la pagina si riferisce; il suo compito è quello di richiamare i JavaBean, creandoli se non sono già presenti nella sessione, inizializzarli settando l'id del docente e passare il controllo alla JSP. Il primo JavaBean si occupa inoltre di istanziare e inizializzare l'EJB InsegnamentiBean passandogli come parametro l'id del docente ottenuto dalla servlet; il secondo fa la stessa cosa ma agendo sull'EJB SpLiberoBean. Dalla JSP vengono poi richiamati entrambi i JavaBean per ottenere atraverso di loro la stringa XML contenente i dati sugli insegnamenti, e quella contenente i dati sullo spazio libero. La trasformazione effettuata attraverso la XSLTaglib è ottenuta facendo uso di due fogli di stile opportuni, che forniscono la presentazione dei dati adatta all'utente generico, e quindi in sola lettura. In questo modo la JSP non contiene elementi scriptlet e la presentation logic viene gestita attraverso il codice Java incapsulato nei JavaBeans e nei Custom tags, mentre la parte grafica è gestita in modo nettamente separato attraverso l'HTML e i tag contenuti nella JSP, e i tag XSL contenuti nel foglio di stile.

Si è deciso di mantenere entrambi i JavaBeans nello scope della sessione, poiché contengono informazioni che possono essere necessarie anche per request successive. Infatti dalla homepage del docente l'utente generico ha due possibilità di rimanere all'interno dell'applicazione (esulano infatti dall'interesse i link esterni): cliccare sul riferimento ad una pagina di materiale didattico posto accanto alla dicitura del relativo insegnamento, o cliccare sul riferimento ad un file da scaricare nella zona dello spazio libero. La prima lo porta alla servlet, denominata richiesta, che si occupa di gestire la parte di applicazione dedicata al materiale didattico di cui si parlerà in seguito; la seconda lo porta alla servlet, denominata download, che effettua il download del file richiesto il cui id viene passato come parametro. Quest'ultima operazione viene effettuata inviando l'id del file all'EJB SpLiberoBean il quale ne restituisce lo stream; entrambi i trasferimenti avvengono attraverso il JavaBean paginasplibero che fornisce la rappresentazione nel View dell'EJB. La servlet,

ottenuto lo stream del file, lo invia sulla response da spedire al client, non prima di aver settato il corretto content type, sempre ricavato dall'EJB.

E' importante sottolineare però che, mentre il riferimento ai due JavaBean permane finchè la sessione utente rimane attiva, il riferimento che questi a loro volta mantengono verso i relativi EJB deve essere mantenuto solo finchè l'utente resta nelle pagine relative allo specifico docente. Quando l'utente entra nell'homepage di un altro docente attraverso la servlet di ingresso hpdoc, vengono istanziati e inizializzati i nuovi EJB passando come parametro il nuovo id docente.

Si osserva inoltre che i JavaBean che vengono utilizzati dalla homepagedocente.jsp, essendo associati all'utente generico, forniscono una visione limitata ai soli metodi di lettura dei corrispondenti EJB.

Si riporta qui di seguito lo schema a blocchi di una prima parte dello scenario dell'utente generico, che comprende l'arrivo della request del client con l'id del docente, l'inizializzazione dei JavaBeans, il recupero dei dati della homepage dal database attraverso l'EJB tier e il passaggio di questi in formato XML al Web tier, e infine la loro visualizzazione attraverso le tag libraries e i fogli di stile.

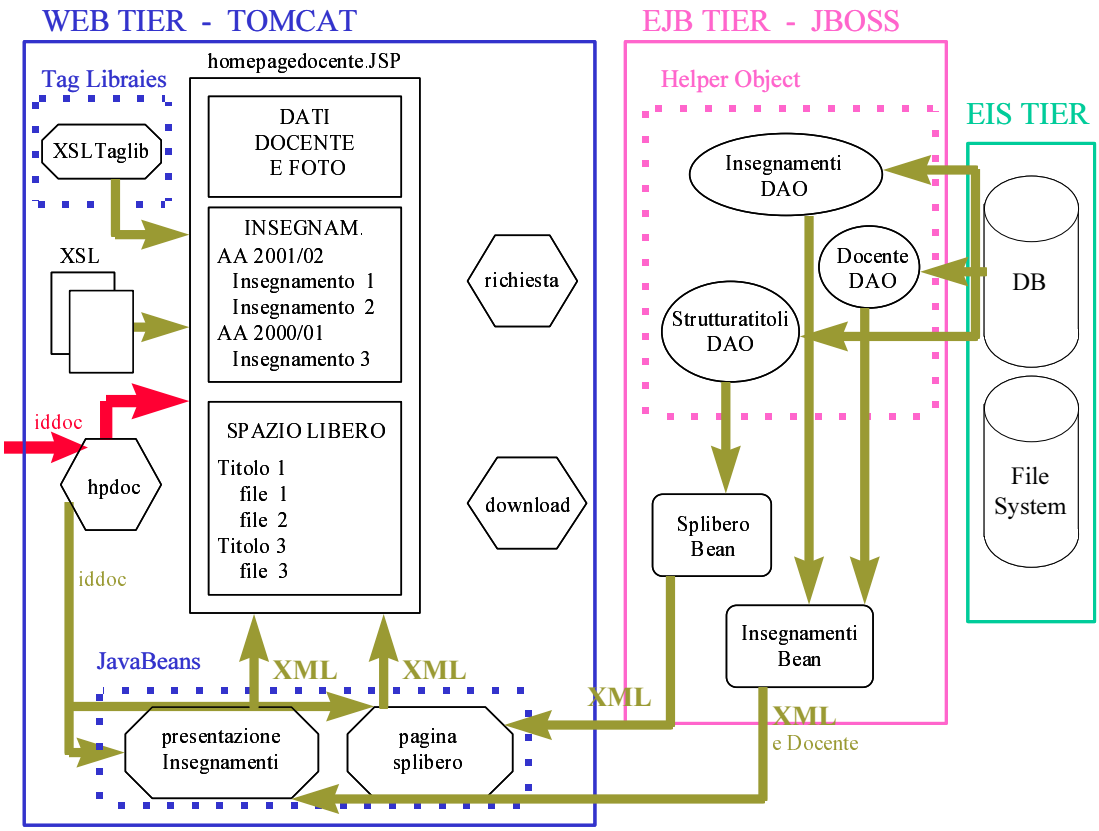


fig 4.15 - Schema a blocchi di parte dello scenario dell'utente generico -

Scenario dell'utente studente

Lo scenario dell'utente studente è quello che si prospetta a partire dalla servlet di nome "richiesta" alla quale si accede, come già detto, cliccando sul link del materiale didattico relativo ad un insegnamento.

Questa servlet si occupa di recuperare il JavaBean paginamatdid che riflette le sole funzionalità di lettura previste dall'EJB MatDidBean. A differenza di quanto veniva fatto dalla servlet hpdoc, la servlet richiesta deve occuparsi anche di stabilire se l'utente è già stato autenticato; nel primo caso il controllo viene passato direttamente alla servlet denominata controlla, che si occupa di stabilire se l'utente autenticato è effettivamente autorizzato ad accedere alla pagina di materiale didattico; nel secondo caso viene passata la request alla JSP matdidlogin.jsp che

contiene il form di login nel quale l'utente può inserire i propri username e password, e poi alla servlet login che non fa altro che inserire tali parametri come attributi dello session object, ed infine alla servlet controlla.

La servlet controlla si occupa di stabilire se i parametri username e password memorizzati nella sessione sono corrispondenti a quelli recuperati dall'EJB attraverso il JavaBean. Ovviamente esiste anche il caso in cui si giunga a questa servlet a sessione scaduta e che quindi non sia possibile recuperare i parametri di autenticazione o il JavaBean o nessuno di essi, nel qual caso si passa alla visualizzazione di un messaggio di errore. Un altro errore verrà comunicato se, pur avendo recuperato i parametri dalla sessione, questi risultano diversi da quelli necessari recuperati dal bean.

Se il processo di autenticazione ha avuto successo si procede passando alla JSP matdid.jsp che richiama il JavaBean paginamatdid e, attraverso di questo, ottiene la stringa XML dall'EJB MatDidBean contenente i dati della pagina di materiale didattico. Come avveniva per l'homepage del docente, anche in questo caso vengono sfruttati la XSLTaglib e un opportuno foglio di stile per visualizzare i dati col layout grafico desiderato. Anche in questo caso la JSP è completamente sgombra da elementi scriptlets e la parte grafica è totalmente separata da quella di codice.

La funzionalità fornita dalla JSP matdid.jsp è assolutamente analoga a quella relativa alla parte di spazio libero già illustrata precedentemente, alla quale è stato aggiunto un link che riporta alla homepage e un link che consente il logout.

Scenario dell'utente docente

Infine rimane da analizzare lo scenario dell'utente docente.

La prima pagina che appare è una JSP con un form di login per l'autenticazione del docente. I parametri username e password inseriti, vengono passati alla servlet di login che li memorizza nella sessione con nomi differenti da quelli eventualmente memorizzati per l'utente studente. Naturalmente esiste la possibilità di differenziare i due processi di memorizzazione grazie al passaggio tramite, un elemento "hidden"

del form di login, di un ulteriore parametro di selezione. La request viene passata poi alla servlet che si occupa di effettuare il controllo dell'autenticazione, denominata controlladoc, la quale crea il JavaBean presentazioneInsegnamenti, se già non esisteva, riinizializzandolo con lo username del docente e prelevandone la password da confrontare con quella fornita. Naturalmente la riinizializzazione del JavaBean implica come al solito la rimozione e ricreazione del corrispondente EJB InsegnamentiBean.

Se l'autenticazione ha avuto successo si procede con la visualizzazione della JSP elencoinsegnamenti.jsp che contiene i dati del docente e l'elenco di tutti gli insegnamenti da lui tenuti negli ultimi tre anni accademici. Questi dati vengono trasformati in formato XML dall'EJB e immessi nella pagina attraverso il corrispondente JavaBean, ma questa volta si utilizza un differente foglio di stile XSL, pochè occorre includere funzionalità aggiuntive che non dovevano essere presenti nella visualizzazione dei medesimi dati da parte dell'utente generico. Infatti l'utente docente ha la possibilità di cliccare su un bottone posto accanto a ciascun insegnamento che gli consente di accedere alla corrispondente pagina di materiale didattico in modifica, e non solo in lettura come avveniva per l'utente generico. Tale bottone porta ad una servlet, denominata docrichiesta, il cui funzionamento è descritto in seguito.

Inoltre nella JSP elencoinsegnamenti.jsp sono presenti, oltre al solito pulsante di logout, anche un pulsante per modificare il file immagine contenente la foto personale del docente, e un altro per accedere in modifica alla pagina di spazio libero. Il primo porta ad una servlet, denominata esegui, il cui compito è quello di eseguire tutte le operazioni di modifica che il docente può effettuare, passando i parametri all'opportuno metodo del JavaBean che a sua volta li trasferisce al corrispondente metodo dell'EJB; il secondo porta invece alla servlet docrichiestaspl, le cui funzionalità verranno descritte in seguito.

Di seguito viene riportato lo schema a blocchi di questa parte dello scenario dell'utente docente, che comprende l'arrivo della request del client al form di login, il

controllo dell'autenticazione del docente, l'inizializzazione dei JavaBeans, il recupero dei dati della pagina elencoinsegnamenti.jsp dal database attraverso l'EJB tier e il passaggio di questi in formato XML al Web tier, e infine la loro visualizzazione attraverso le tag libraries e i fogli di stile. I fogli di stile in questo caso fanno comparire anche i bottoni per le modifiche.

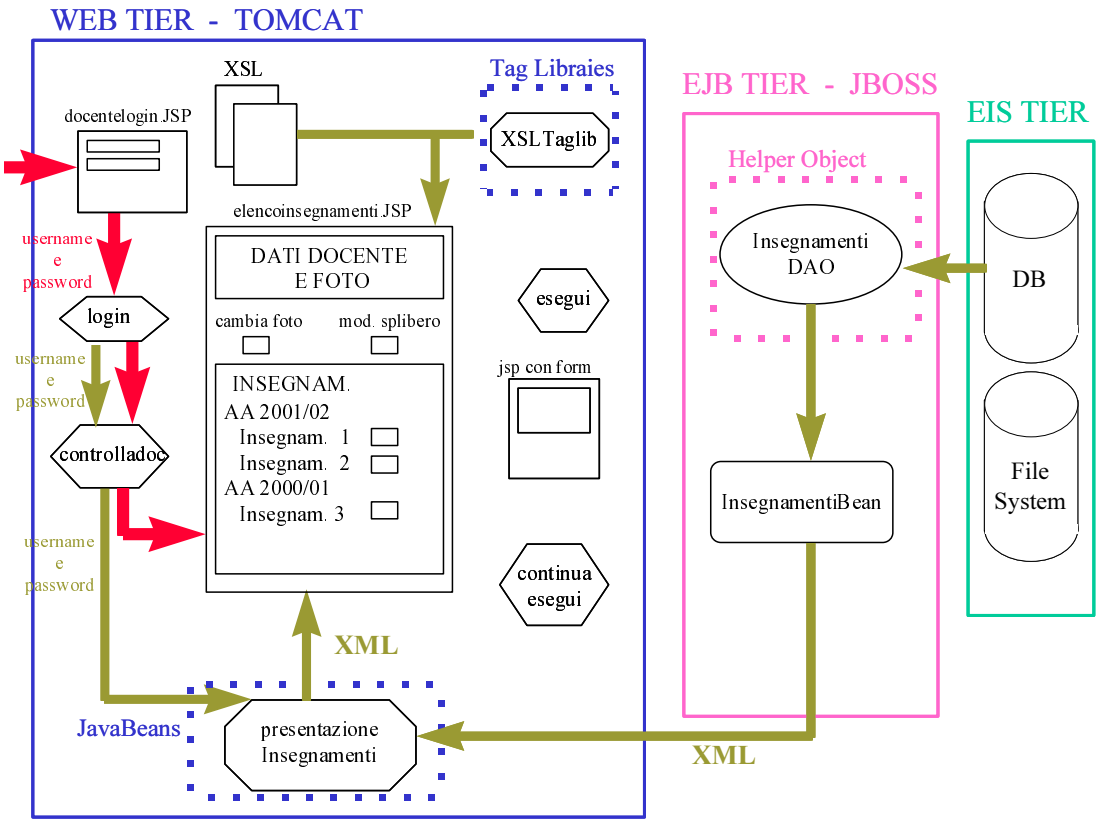


fig 4.16 - Schema a blocchi di parte dello scenario dell'utente docente -

Si analizza ora lo scenario di modifica della pagina di materiale didattico relativa ad un insegnamento, al quale si accede attraverso la servlet docrichiesta. Questa effettua la creazione e inizializzazione del JavaBean presentazionePagina che fornisce la visione completa delle funzionalità dell'EJB MatDidBean. Viene fatto anche un ulteriore controllo sul diritto di accesso dell'utente docente sulla pagina

richiesta, poiché egli può accedere in modifica alle sole pagine di materiale didattico relative ad insegnamenti di sua competenza.

Se l'esito dei controlli è positivo viene visualizzata la JSP matdiddocente.jsp che, ricavando i dati in formato XML, li visualizza utilizzando un foglio di stile che aggiunge tutte le funzionalità di modifica previste. Queste comprendono bottoni di cancellazione, aggiornamento e aggiunta di ciascuna voce presente nella struttura della pagina, quindi titoli, files , note e links. Tutte queste funzionalità vengono gestite attraverso una servlet, denominata esegui, alla quale vengono passati tutti i parametri del caso, compresi vari parametri di selezione sull'azione da eseguire. Questa servlet si occuperà di effettuare direttamente le operazioni richieste, se queste non necessitano di un passaggio intermedio attraverso un form opporuno contenuto in una JSP, cosa che viene gestita invece da un'altra servlet, denominata continuaesegui, che si occuperà di effettuare le modifiche.

La servlet continuaesegui si occupa anche di ricevere le request multipart/form-data che consentono l'upload dei files. Facendo uso delle librerie O'Reilly già precedentemente citate, essa provvede a identificare ciascuna parte della request rispedendole all'EJB, e salva il file nella opportuna directory.

L'altra funzione disponibile al docente è quella di cambiamento dello username e della password che l'utente studente deve utilizzare per accedere alla pagina in lettura, cosa che viene gestita sempre dalla servlet esegui dopo che il docente ha immesso i nuovi valori dei parametri di autenticazione attraverso una JSP denominata cambiapassword.jsp.

Una volta completata l'operazione da eseguire si ritorna alla matdiddocente.jsp che recupera i dati in formato XML aggiornati dall'EJB.

Si riporta il solito schema a blocchi della parte di scenario dell'utente docente in cui si presume sia già stato scelto e visualizzato il materiale didattico di un insegnamento, e viene effettuata l'operazione di upload di un file da parte del docente.

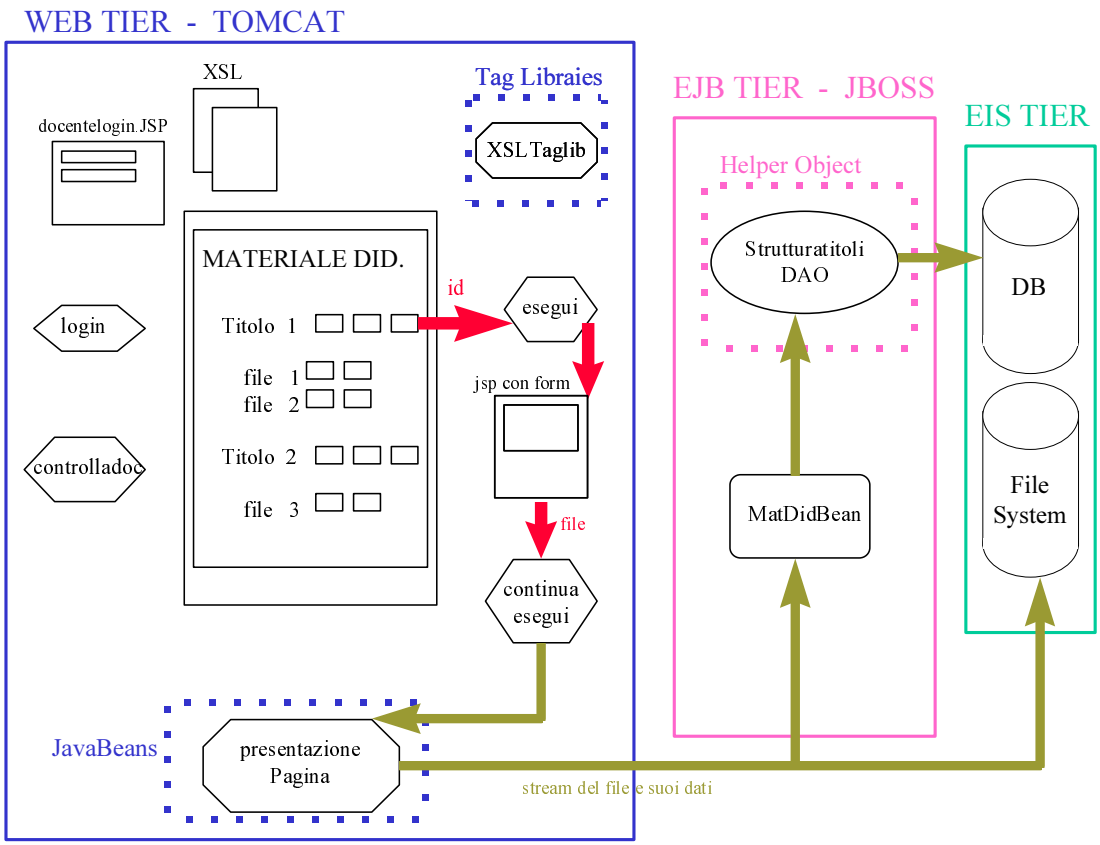


fig 4.17 - Schema a blocchi dell'upload di un file da parte dell'utente docente -

Infine l'ultima parte dello scenario dell'utente docente è quella di aggiornamento della pagina di spazio libero. Le funzionalità sono del tutto analoghe a quelle già illustrate per le pagine di materiale didattico, con l'esclusione del cambio della password, poiché non è richiesta alcuna autenticazione per l'utente che intende accedere in lettura alla parte di spazio libero.

A causa della forte analogia con lo schema precedente, quest'ultimo scenario non viene ulteriormente illustrato.

4.3.3 Controller

Gli oggetti rientranti nel ruolo di Controller sono stati realizzati come componenti web, e sono le servlet esegui e continuaesegui. Esse infatti hanno un ruolo di mediazione traducendo la richiesta dell'utente che agisce sui bottoni presentati dai View-objects, in opportuni "eventi" che scatenano l'invocazione dei corrispondenti metodi di aggiornamento dei Model-objects.

Questi eventi altro non sono che, a loro volta, l'invocazione di metodi speculari dei JavaBeans corrispondenti agli EJB coinvolti nell'operazione. Questi JavaBeans, hanno il ruolo fondamentale di mantenere il riferimento all'EJB corrispondente attraverso richieste multiple dell'utente, e ne gestiscono il ciclo di vita, rimuovendolo se non più necessario, ricreandolo e rinizializzandolo se richiesta una nuova istanza.

Infine compito delle servlet Controller è anche quello di passare il controllo all'opportuna JSP una volta terminata l'operazione di aggiornamento. Infatti, essendo le operazioni di modifica sulla pagina di spazio libero analoghe a quelle delle varie pagine di materiale didattico, la servlet che ne gestisce l'esecuzione è sempre la stessa, ma la JSP da visualizzare al termine è ovviamente diversa nei due casi. Anche il riferimento al JavaBean è diverso, nel primo caso è una istanza di paginasplibero, nel secondo è una istanza di paginamatdid. Questo problema è stato superato introducendo una superclasse astratta, denominata paginapresentazione, che può essere riferita dalla servlet indipendentemente dal tipo di bean. Ciò ovviamente è stato possibile grazie al fatto che entrambi i bean hanno le stesse funzionalità, ma si riferiscono EJB diversi (SpLiberoBean nel primo caso, MatDidBean nel secondo). Nell'EJB-tier è stata fatta la stessa cosa con gli EJB avendo introdotto, come già menzionato, la superclasse Strutturatitoli.

4.4 Alcune classi Java realizzate

Per ragioni di spazio e di semplicità verranno in questa sede riportate solo alcune delle classi Java realizzate, scegliendo tra esse quelle più significative.

4.4.1 Classi dell'EJB tier

Nell'EJB tier le classi sono state organizzate in package e questi raggruppati in alcuni moduli JAR:

1) util.jar contenente alcune classi di utilità generale all'EJB tier tra le quali le eccezioni personalizzate organizzate nel package "eccezioni", il VO Docente, i DAO e altre superclassi fondamentali organizzate nel package "helper";
2) insegnamenti.jar contenente la home e la remote interface, e la classe dell'EJB che realizza l'oggetto Insegnamenti;
3) matdid.jar contenente la home e la remote interface, e la classe dell'EJB che realizza l'oggetto Materiale Didattico;
4) splibero.jar contenente la home e la remote interface, e la classe dell'EJB che realizza l'oggetto Spazio Libero;

Classe astratta ContenutoDOM

La prima classe che è stata implementata nel EJB tier è stata quella che definisce i metodi di utilizzo della struttura DOM comuni a tutti gli EJB realizzati.

Essa ha una variabile d'istanza di tipo Document che mantiene la struttura DOM ed è stata definita astratta a causa della presenza del metodo Carica che si occupa di caricare la struttura con i dati del database e quindi deve essere implementato in modo specifico dalle sottoclassi.

```
package helper;
```

```
import java.rmi.RemoteException;

// API JAXP-DOM
import javax.xml.parsers.*;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;
import java.util.*;

import eccezioni.*;

public abstract class ContenutoDOM  {

  // variabile di istanza che contiene il documento DOM
  protected Document doc = null;

 //  ### METODI DI RESTITUZIONE DEL RISULTATO IN
FORMATO XML ###


/* Metodo che restituisce il risultato sotto forma di
stringa XML */

public ByteArrayOutputStream getDOM() throws
RemoteException {

  ByteArrayOutputStream os = new ByteArrayOutputStream();
  try {
      StreamResult result = new StreamResult(os);
      TransformerFactory tFactory =
                  TransformerFactory.newInstance();
       Transformer transformer =
                     tFactory.newTransformer();
       DOMSource source = new DOMSource(doc);
       transformer.transform(source, result);

  } catch (TransformerConfigurationException tce) {
```

```
    // Errore generato dal parser
    throw new RemoteException("Errore nella trasformazione
del DOM
          in stringa XML");
  } catch (TransformerException te) {
     // Errore generato dal parser
    throw new RemoteException("Errore nella trasformazione
del DOM
          in stringa XML");
  }

  return os;
}


  //    ### METODI DI COSTRUZIONE DOM ####

  // metodo che costruisce la struttura DOM caricandola
con i dati opportuni

   public abstract void Carica() throws RemoteException ;


} // fine classe ContenutoDOM
```

La classe Strutturatitoli

Questa classe eredita da ContenutoDOM e a sua volta è superclasse degli EJB del materiale didattico e dello spazio libero che presentano forti analogie proprio a causa della particolare struttura DOM che mantengono.

Come variabile di istanza viene mantenuto un oggetto DAO di tipo StrutturatitoliDAO che incapsula tutti gli statement SQL che agiscono su materiale didattico e spazio libero.

Essendo molto lunga vengono riportati solo alcuni metodi come ad esempio quello di cancellazione di un file. Si noti la presenza delle tre operazioni in sequenza:

aggiornamento nel database, aggiornamento nella struttura DOM e aggiornamento nel File System.

Per meglio illustrare il funzionamento e l'utilizzo delle API JAXP che sfruttano la struttura DOM viene riportato anche un metodo privato che crea un nuovo nodo di tipo file alla struttura; l'elemento ritornato può così essere aggiunto come figlio ad un altro con il metodo `appendChild`.

Si riporta infine l'implementazione del metodo astratto Carica ereditato da ContenutoDOM che realizza il caricamento della struttura ricavandone i dati dal database tramite il DAO.

```
package helper;


// API JAXP-DOM
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.parsers.*;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.*;

import java.io.*;
import java.util.Hashtable;
import java.util.Enumeration;

import helper.*;

import eccezioni.*;

public class Strutturatitoli extends ContenutoDOM {

  protected String path = null;
  protected int maxidt = 0;
  protected int maxidn = 0;
  protected int maxidf = 0;
```

```
  protected int maxidl = 0;
  protected boolean scrivere = false;

  // riferimento al DAO
  protected StrutturatitoliDAO Dao = new
StrutturatitoliDAO();


  // .....


 private Element NuovoNodoFile(String idf, String nomefile,
String contenttype, String descrizione, int ordine)  {

  // crea il nuovo nodo file
  Element newFile = doc.createElement("file");

  // crea lÆattributo nome
  Attr nomeattr = doc.createAttribute("nome");

  // setta l'attributo nome
  nomeattr.setValue(nomefile);

  newFile.setAttributeNode(nomeattr);

  // crea lÆatributo contenttype
  Attr contenttypeattr = doc.createAttribute("contenttype");

  // setta l'attributo contenttype
  contenttypeattr.setValue(contenttype);

  newFile.setAttributeNode(contenttypeattr);

  // crea la descrizione
  Text textDesc = doc.createTextNode(descrizione);
  newFile.appendChild(textDesc);

  // AGGIUNTA DELL'ATTRIBUTO ID
  Attr idattr = doc.createAttribute("id");
  idattr.setValue(new Integer(maxidf).toString());
  newFile.setAttributeNode(idattr);
  maxidf ++;
```

```
  // AGGIUNTA DELL'ATTRIBUTO IDF
  Attr idfattr = doc.createAttribute("idf");

  // setta l'attributo nome al valore passato come
parametro
  idfattr.setValue(idf);
  newFile.setAttributeNode(idfattr);

  // AGGIUNTA DELL'ATTRIBUTO ORDINE
  Attr ordineattr = doc.createAttribute("ordine");
  ordineattr.setValue(new Integer(ordine).toString());
  newFile.setAttributeNode(ordineattr);

  return newFile;

}

// .....

public void EliminaFile(String idfile) throws
RemoteException {

    try {
         Element f = TrovaNodo("file",idfile);

         String nomefile = f.getAttribute("nome");

         // CANCELLAZIONE DAL DATABASE TRAMITE DAO

         // ritrova l'id titolo relativo al DAO
      String idtrel =
((Element)f.getParentNode()).getAttribute("idt");

         // ritrova l'id file relativo al DAO
         String idfrel = f.getAttribute("idf");

         Dao.cancFile(idfrel,idtrel);

         // CANCELLAZIONE DALLA STRUTTURA DOM

         // rimuove il nodo dal DOM
```

```
        ((Element)f.getParentNode()).removeChild(f);

        // CANCELLAZIONE DAL FILE SYSTEM

        File file = new File(path + File.separator +
nomefile);

      /* il metodo isFile testa l'esistenza e anche se il
file e' ordinario
       il metodo delete() restituisce vero se e solo se la
cancellazione del file e' avvenuta correttamente */

      if(file.exists()) {

         if (!file.isFile() ) { /* si sta tentando di
cancellare una directory */

              throw new RemoteException("Il file
specificato non e' corretto");
         } else if(!file.delete()) {

              throw new RemoteException("La cancellazione
del file non e' avvenuta correttamente dal server");
         }

      }

      /* se il file specificato non esiste non si fa nulla.
In tal modo si da' la possibilta' di cancellare dal database
i dati di un file che per qualche ragione e' gia' stato
cancellato dal file system */

    }  catch(DOMException nte) {

     throw new RemoteException("La cancellazione del file
non e' avvenuta correttamente dalla struttura DOM perche'
non e' stato trovato il titolo di appartenenza");

    }  catch(DAOException sqle) {

     throw new RemoteException("La cancellazione del file
dal database non " +                    "e' avvenuta
```

```
correttamente a causa del seguente errore: " +
sqle.getMessage());
    }

  }

  // .....

public void Carica() throws RemoteException {

  // inizializza i valori max degli id
  maxidt = 1;
  maxidf = 1;
  maxidl = 1;
  maxidn = 1;

  try {
      Element root = doc.getDocumentElement();

      // ottiene la tabella dei titoli
      Hashtable titoli = Dao.getTitoli();

      // recupera gli attributi dei titoli
      for (Enumeration et = titoli.keys() ;
et.hasMoreElements() ;)
       {

        Integer idt = (Integer)et.nextElement();
        Hashtable rigatitolo = (Hashtable)titoli.get(idt);

        // recupero la denominazione del titolo
        String den =
(String)rigatitolo.get("denominazione");

        // recupero il numero d'ordine del titolo
        Integer ord = (Integer)rigatitolo.get("ordine");
        int o = ord.intValue();

        // crea un nuovo nodo titolo
        Element newTitolo = NuovoNodoTitolo(idt.toString(),
den, o);
```

```
        // aggiunge il titolo alla pagina
        root.appendChild(newTitolo);

        // recupera la tabella dei files di questo titolo
        Hashtable files = Dao.getFiles(idt.toString());
        for (Enumeration ef = files.keys() ;
ef.hasMoreElements() ;)
        {
          Integer idf = (Integer)ef.nextElement();

          /* recupera la riga contenente nome, descrizione
e contenttype del file */

          Hashtable riga = (Hashtable)files.get(idf);
          String nomefile = (String)riga.get("nomefile");
          String contenttype =
(String)riga.get("contenttype");
          String filedescrizione =
             (String)riga.get("filedescrizione");
          Integer ordfile = (Integer)riga.get("ordine");

          // crea un nuovo nodo file
          Element newFile = NuovoNodoFile(idf.toString(),
nomefile, contenttype, filedescrizione, ordfile.intValue());
          newTitolo.appendChild(newFile);

        } // fine iterazione sui files

        // recupera la tabella dei links di questo titolo
        Hashtable links = Dao.getLinks(idt.toString());
        for (Enumeration el = links.keys() ;
el.hasMoreElements() ;)
        {
         Integer idl = (Integer)el.nextElement();

         // recupera la riga contenente nome e descrizione
del link
         Hashtable riga = (Hashtable)links.get(idl);
         String url = (String)riga.get("url");
         String urldescrizione =
(String)riga.get("urldescrizione");
         Integer ordlink = (Integer)riga.get("ordine");
```

```
          // crea un nuovo nodo link
          Element newLink = NuovoNodoLink(idl.toString(),
url, urldescrizione, ordlink.intValue());
          newTitolo.appendChild(newLink);

          } // fine iterazione sui links

         // recupera la tabella delle note di questo titolo
         Hashtable note = Dao.getNote(idt.toString());
         for (Enumeration en = note.keys() ;
en.hasMoreElements() ;)
         {
           Integer idn = (Integer)en.nextElement();
           /* recupera la riga contenente nome e
descrizione della nota */
           Hashtable riga = (Hashtable)note.get(idn);
           String notadescrizione =
             (String)riga.get("notadescrizione");
           Integer ordnota = (Integer)riga.get("ordine");

           // crea un nuovo nodo nota
           Element newNota = NuovoNodoNota(idn.toString(),
notadescrizione, ordnota.intValue());
           newTitolo.appendChild(newNota);
}

         } // fine iterazione sulle note

       } catch (Exception de) {
System.out.println("ERRORE:" + de.getMessage());
            throw new RemoteException("Non e' stato
possibile recuperare correttamente i dati dal database: " +
de.getMessage());

    }

} // fine classe Strutturatitoli
```

Il VO Docente

Per illustrare l'utilizzo dei VO si riporta l'unico VO implementato. Come si è già detto questa è una classe serializzabile che ha una struttura molto semplice con metodi getter e setter. Per ragioni di brevità vengono riportati solo i metodi getter e setter di una delle property.

```
package helper;

public class Docente implements java.io.Serializable {

  private String iddocente;
  private String nome;
  private String cognome;
  private String telefono;
  private String email;
  private String ruolo;
  private String ricevimento;
  private String fax;
  private String username;
  private String password;
  private String fotofile;
  private String fototype;
  private String fotopath;
  private String ricerca;
  private String cooperazioni;
  private String incarichi;
  private String affiliazioni;
  private String collaborazioni;
  private String ufficio;
  private String pathsplibero;
  private String homepage;

  // costruttore

  public Docente() {

  }
```

```
  // metodi getter
  public int getId() {
     return Integer.parseInt(iddocente);
  }
// ...

  // metodi setter
  public void setId(int i) {
     iddocente = new Integer(i).toString();
  }
} // fine classe Docente
```

I DAO

Per rendere l'idea di come sono stati implementati i DAO per incapsulare gli statement SQL si riporta il più semplice, quello relativo al VO Docente.

Si noti l'uso delle API JNDI nel metodo getDataSource per recuperare il DataSource dal namespace.

```
package helper;
import javax.naming.*;
import javax.naming.directory.*;
import java.sql.*; import javax.sql.*;

import eccezioni.*;
```

```
public class DocenteDAO implements java.io.Serializable{

   private int iddocente = 0;
   private String username = null;

   private StringBuffer td = new StringBuffer("SELECT *
FROM TBL_DOCENTI");

   static String nomeJNDI = "java:/SQLServerPool";

   private static DataSource getDataSource() throws
        DatasourceNonTrovatoException {

     try {
InitialContext ic = new InitialContext();
return (DataSource) ic.lookup(nomeJNDI);

     } catch (NamingException ne) {

             throw new DatasourceNonTrovatoException("pool
di connessioni al database non trovato");
     }
   }

  public DocenteDAO(int idd) {

    String s = new Integer(idd).toString();
    td.append(" WHERE Id= " + s);

  }

  public DocenteDAO(String u) {

    td.append(" WHERE Username = '" + u + "'");
  }

//            ### *** METODI DI SELEZIONE ***###

  public Docente getDati() throws DAOException {

     Docente doc = new Docente();
```

```
      Connection con = null;
      Statement stmdocente = null;
      ResultSet docente = null;
      try {
          con = getDataSource().getConnection();
          stmdocente = con.createStatement();
          docente = stmdocente.executeQuery(td.toString());

          // costruzione della tabella hash degli
insegnamenti del docente iddocente per l'anno idanno

          if(docente.next()) { // docente riconosciuto

        // riempie l'istanza docente con i dati estratti dal
database

          doc.setId(docente.getInt("Id"));
          doc.setNome(docente.getString("Nome"));
          doc.setCognome(docente.getString("Cognome"));
          doc.setUsername(docente.getString("Username"));
          doc.setPassword(docente.getString("Passwd"));
          doc.setTelefono(docente.getString("Telefono"));
          doc.setEmail(docente.getString("Email"));
          doc.setFax(docente.getString("Fax"));
          doc.setRuolo(docente.getString("Ruolo"));
          doc.setFotopath(docente.getString("FotoPath"));
          doc.setFotofile(docente.getString("FotoFile"));
          doc.setFototype(docente.getString("FotoType"));
          doc.setUfficio(docente.getString("Ufficio"));

doc.setRicevimento(docente.getString("Ricevimento"));

doc.setPathsplibero(docente.getString("PathSplibero"));
          doc.setHomepage(docente.getString("HomePage"));

          } else { // docente non trovato

           doc = null;

          }

          docente.close();
```

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

      } catch(Exception e) {
          throw new DAOException(e.getMessage());
      }

      return doc;

    }

   public void nuovafoto(String u, String nome, String
fototype) throws DAOException {

       Connection con = null;
       PreparedStatement stmfoto = null;
       try {
         con = getDataSource().getConnection();
         String nfoto = "UPDATE TBL_DOCENTI" +
                        " SET FotoFile = ? ," +
                        "     FotoType = ? " +
                        " WHERE Username = ? ";

         stmfoto = con.prepareStatement(nfoto);
         stmfoto.setString(1, nome);
         stmfoto.setString(2, fototype);
         stmfoto.setString(3, u);

         int nr = stmfoto.executeUpdate();
         stmfoto.close();
         con.close();
         if (nr == 0) {
             throw new DAOException("Lo statement SQL di
inserimento non ha inserito alcuna riga nella tabella dei
titoli");
         }

         } catch(Exception e) {
            throw new DAOException(e.getMessage());
}

         }
```

```
    }

}  // fine classe DocenteDAO
```

L'EJB del materiale didattico

Si riporta ora nell'ordine la home interface, la remote interface e la classe dell'EJB che implementa il materiale didattico di un insegnamento.

Home interface:

```
package ejb.MatDid;

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
import ejb.MatDid.MatDidEJB;

public interface MatDidHome extends EJBHome {

    MatDidEJB create(String idcorso, String idanno, String
idinsegnamento) throws RemoteException, CreateException;

    void remove() throws RemoteException;

} // fine della remote interface
```

Remote interface:

```
package ejb.MatDid;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

import java.io.*;

import org.w3c.dom.Document;
```

```
import org.w3c.dom.DOMException;

public interface MatDidEJB extends EJBObject {


   // metodi di utilizzo dei soli docenti

   public void NuovoTitolo(String testo) throws
RemoteException;

   public void NuovoFile(String idtitolo, String nomefile,
String contentype, String desc, boolean nuovo) throws
RemoteException;

   public boolean FileEsistente(String nomefile) throws
RemoteException;

   public void NuovoLink(String idtitolo , String url , String
descrizione) throws RemoteException;
   public void NuovaNota(String idtitolo , String
descrizione) throws RemoteException;

   public void EliminaTitolo(String idtitolo) throws
RemoteException;
   public void EliminaFile(String idfile) throws
RemoteException;
   public void EliminaLink(String idlink) throws
RemoteException;
   public void EliminaNota(String idnota) throws
RemoteException;

   public void ModificaTitolo(String idtitolo, String testo)
throws RemoteException;
   public void ModificaFile(String idfile, String descrizione)
throws RemoteException;
   public void ModificaLink(String idlink , String url ,
String descrizione) throws RemoteException;
   public void ModificaNota(String idnota , String
descrizione) throws RemoteException;
   public void Scambia(String tipo , String id , String
verso) throws RemoteException;
   public void Cambiapub() throws RemoteException;
```

```
   public void CambiaPw(String newusername, String
newpassword) throws RemoteException;
   public void Ripristinadefault() throws RemoteException ;


   public String getXmlfile() throws RemoteException;
   public void setXmlfile(String name) throws
RemoteException;

   public String getTitolo(String idfile) throws
RemoteException;
   public String getNomefile(String idfile) throws
RemoteException;
   public String getContenttype(String idfile) throws
RemoteException;
   public String getFiledescrizione(String idfile) throws
RemoteException;
   public String getUrl(String idlink) throws
RemoteException;
   public String getUrldescrizione(String idlink) throws
RemoteException;
   public String getNota(String idnota) throws
RemoteException;
   public String getPath() throws RemoteException;
   public String getPub() throws RemoteException;

   public boolean accessoDoc(String username, String
password) throws RemoteException;

   // metodi di utilizzo anche degli studenti

   public ByteArrayOutputStream getDOM() throws
RemoteException;

   public FileInputStream downloadfile(String idfile) throws
RemoteException;

   public String getCorsodesc() throws RemoteException;
   public String getAnnodesc() throws RemoteException;
   public String getInsegnamentodesc() throws
RemoteException;
```

```
   public String getUsername() throws RemoteException ;
   public String getPassword() throws RemoteException ;
   public short getDefaultflag() throws RemoteException ;
   public String getUsernamedefault() throws
RemoteException ;
   public String getPassworddefault() throws
RemoteException ;

   public void Carica() throws RemoteException;


} // fine remote interface
```

La classe del bean eredita da Strutturatitoli la maggior parte dei metodi che sono definiti nella remote interface. Restano da implementare quelli più specifici come ad esempio quello che recupera la denominazione di un insegnamento facendo uso del DAO, o quello che modifica la password.

```
package ejb.MatDid;

import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import java.util.Hashtable;
import java.util.Enumeration;

import helper.*;
import eccezioni.*;

public class MatDidBean extends Strutturatitoli implements
SessionBean {

  private String idcorso = null;
  private String idanno = null;
  private String idinsegnamento = null;
  private String corsodesc = null;
  private String annodesc = null;
```

```
  private String insegnamentodesc = null;
  private String username = null;
  private String password = null;
  private short df;
  private String usernamedefault;
  private String passworddefault;

  public String getCorsodesc()  {        return corsodesc;
    }

   public String getAnnodesc()  {        return annodesc;
    }
   public String getInsegnamentodesc()  {        return
insegnamentodesc;
    }

// ...

public void CambiaPw(String newusername, String
newpassword) throws RemoteException {

      try{
       Dao.cambiapassword(newusername, newpassword);
       username = newusername;
       password = newpassword;
       df = 0;

       }  catch(DAOException daoe) {
           throw new RemoteException(daoe.getMessage());
       }

}


public void ejbCreate(String idc ,String ida, String idi)
throws RemoteException {

    //inizializza parametri
   idcorso = idc;
   idanno = ida;
   idinsegnamento = idi;
```

```
   // inizializza il DAO
   Dao.setMatDidDAO(idcorso,idanno,idinsegnamento);

   try {
     // crea la radice
                Crearoot();

                 // inizializza le descrizioni
                 corsodesc = Dao.getCorsoDesc();
                 annodesc = Dao.getAnnoDesc();
                 insegnamentodesc =
Dao.getInsegnamentoDesc();
                 path = Dao.getPath();
                 System.out.println("path = " + path);


               // recupera username e password e
defaultflag

                 Hashtable dati = Dao.getUsername();
                username =
((String)dati.get("username")).trim();
                 password =
((String)dati.get("password")).trim();
                 df =
((Short)dati.get("defaultflag")).shortValue();
                 dati = Dao.getUsernamedefault();
                 usernamedefault =
((String)dati.get("usernamedef")).trim();
                 passworddefault =
((String)dati.get("passworddef")).trim();


    } catch (Exception de) {

        throw new RemoteException(de.getMessage());
    }

} // fine classe del EJB
```

4.4.2 Classi del Web tier

Nel Web tier si è creato un modulo Web di nome matdid.war contenente i seguenti componenti:

1) files JSP, files di immagini e fogli di stile (file .xsl) opportunamente collocati in apposite directory a seconda della loro funzione;
2) classi Java delle servlet collocate nella directory /WEB-INF/class e dei JavaBeans collocati nella directory /WEB-INF/class/bean
3) librerie di varia natura, comprese quelle per l'upload dei files e dei custom tag XSLTaglibs collocate nella directory /WEB-INF/lib.

Si riporta il codice solo di alcune JSP e alcune servlet tra le più significative.

Codice della JSP dell'homepage di un docente

Si riporta a titolo esemplificativo la JSP realizzata per la homepage di un docente secondo la visione dell'utente generico.

Da notare in particolare i seguenti punti principali:

1) il meccanismo delle inclusioni utilizzato per mantenere un template comune a tutte le pagine, con una parte alta che riporta il logo dell'università e una fila verticale di bottoni sulla sinistra della pagina;
2) l'utilizzo di due custom tags: uno per l'utilizzo di XML e fogli di stile (tag <xsl:apply>) e l'altro per il controllo del flusso (tag <jLib:if>);
3) l'utilizzo di due JavaBeans con scope session che, come già detto più volte, fanno riferimento agli EJB opportuni
4) i dati del docente sono ricavati con opportuni tag <jsp:getProperty> che agiscono sul bean degli insegnamenti che a sua volta li ricava dal VO Docente.

```
<%@taglib uri="http://jakarta.apache.org/taglibs/xsl-1.0"
prefix="xsl" %>

<%@ taglib uri="http://jakarta.apache.org/taglibs/utility"
prefix="jLib" %>

<jsp:useBean id="ins"
class="bean.presentazioneInsegnamenti" scope="session"/>


<!-- esegue il controllo -->

<% ins.Controlladoc(); %>


<jLib:If predicate='<%=!ins.getOk()%>'>

        <jsp:forward page="/error2.jsp">
            <jsp:param name="msg" value="Docente non
trovato"/>
            <jsp:param name="pagrit" value="/md/"/>
        </jsp:forward>
</jLib:If>

<jsp:useBean id="spl" class="bean.paginasplibero"
scope="session"/>

<html>

<head>

  <title>University of Modena e Reggio Emilia - Facolta di
Ingegneria  Modena</title>

  <style type="text/css">

   </style>

</head>
```

```
<body text="#000000" bgcolor="#FFFFFF" link="176EA5"
alink="176EA5" vlink="176EA5">

<center>

<!-- inclusione parte alta -->

<jsp:include page="/partealta.html"/>

<!-- fine parte alta  ----------   ----------
---------- --- -->

<br>
<br>

<table width="750" border=0 cellpadding="2"
cellspacing="0">

<tr>
     <!--  colonna coi bottoni -->
     <jsp:include page="/bottoni.html"/>

<!--  colonna con le informazioni -->
      <td valign="top">

     <center>

     <table width=550 border=0 cellpadding="2"
cellspacing="0">

     <!--  riga con il nome e foto del docente -->
     <tr>
       <td align="center">

       <table width="559" border="0" cellpadding="5">

       <tr>
         <td>

           <font face="arial narrow" size="4">
```

```
            <jsp:getProperty name="ins"
property="nome"/> <jsp:getProperty name="ins"
property="cognome"/>
          </font>

        </td>

        <td>  <!-- foto -->

           <jLib:If predicate='<%= ins.okfoto() %>'>

              <img src="<%= ins.getCont("stu") %>"
width="100" heigh="300" border="0" alt="Foto"></img>

           </jLib:If>

           <jLib:If predicate='<%= !ins.okfoto() %>'>
               
           </jLib:If>

        </td>
      </tr>

      </table>

      </td>
    </tr>

       <!--  riga con i dati del docente -->

    <tr>
      <td align="right">

        <table width="559" border="0" cellpadding="5">

           <tr valign="top">
        <td>

          <font face="arial narrow"
size="3"><b>Ufficio:</b></font>
        </td>
        <td>
```

```
            <font face="arial narrow"
size="3">DSI<br><jsp:getProperty name="ins"
property="ufficio"/></font>
          </td>
          </tr>

          <tr valign="top">
          <td>
            <font face="arial narrow"
size="3"><b>Tel.:</b></font>
          </td>
          <td>
            <font face="arial narrow"
size="3"><jsp:getProperty name="ins"
property="telefono"/></font>
          </td>
          </tr>

          <tr valign="top">
          <td>
            <font face="arial narrow"
size="3"><b>Fax:</b></font>
          </td>
             <td>
               <font face="arial narrow"
size="3"><jsp:getProperty name="ins"
property="fax"/></font>
          </td>
          </tr>

          <tr valign="top">
             <td>
            <font face="arial narrow" size="3"><b>E-
mail:</b></font>
          </td>
          <td><a HREF="mailto:<%=ins.getEmail()%>">
               <font face="arial narrow"
size="3"><jsp:getProperty name="ins"
property="email"/></font>
             </a>
          </td>
```

```
          </tr>


          <tr valign="top">
          <td>
            <font face="arial narrow"
size="3"><b>Ricevimento:</b></font>
          </td>
          <td>
            <font face="arial narrow"
size="3"><jsp:getProperty name="ins"
property="ricevimento"/></font>
          </td>
          </tr>

          <tr valign="top">
          <td>
            <font face="arial narrow"
size="3"><b>Homepage personale:</b></font>
          </td>
          <td>
            <font face="arial narrow" size="3"><a
href="<%= ins.getHomepage() %>"><jsp:getProperty
name="ins" property="homepage"/></a></font>
          </td>
          </tr>

          </table>

        </td>
      </tr> <!-- fine riga con i dati -->

      <!--  riga con gli insegnamenti -->
      <tr>
         <td align="center">
         <!-- effettua prima il controllo -->

         <% ins.Controllapag(); %>

         <jLib:If predicate='<%= !ins.getOk() %>'>
```

```
          <big>Non e' stato possibile trovare gli
insegnamenti del docente</big>
          <br>
             <big>
                  <jsp:getProperty name="ins"
property="msg"/>
             </big>

           </jLib:If>


        <jLib:If predicate='<%= ins.getOk() %>'>


          <xsl:apply nameXml="ins"
propertyXml="paginaxml" xsl="/xml/insegnamenti.xsl"/>

        </jLib:If>

        </td>
     </tr>

     <tr> <!-- riga vuota -->

        <td> </td>

     </tr>

     <!-- riga con lo spazio libero -->

     <tr>

        <td align="center">

        <!-- effettua prima il controllo -->

        <% spl.Controllapag(); %>

        <jLib:If predicate='<%= !spl.getOk() %>'>

          <big>Non e' stato possibile recuperare
correttamente la parte di spazio libero del docente</big>
```

```
          <br>
             <big>
                  <jsp:getProperty name="spl"
property="msg"/>
             </big>

           </jLib:If>

         <jLib:If predicate='<%= spl.getOk() %>'>

           <xsl:apply nameXml="spl"
propertyXml="paginaxml" xsl="/xml/spliberoguest.xsl"/>

         </jLib:If>

         </td>

      </tr>

      <!-- riga del logout solo se necessaria -->

      <jLib:If predicate='<%= ins.loggato(session) %>'>

      <tr>
      <td>
          Sei attualmente loggato con username <%=
ins.getUsernamestudente(session) %>

      </td>

      <td>
           <a href="/md/finesessione?st=stu">Log out</a>

      </td>
      </tr>

      </jLib:If>

      </table>

       <!--  fine colonna con le informazioni -->
```

```
        </td>
</tr>
</table>
</body>
</html>
```

Codice della servlet continuaesegui e download

Tra tutte le servlet realizzate quelle senz'altro più significative sono quelle che realizzano il download e l'upload dei files.

Sempre per motivi di brevità si riporta solo il frammento di codice che realizza queste operazioni.

Il download è stato realizzato utilizzando l'OutputStream associato ad una response che consente di inviare via HTTP un qualunque stream di byte. Al fine di consentire al browser del client un immediato riconoscimento del tipo di file inviato è necessario settare l'opportuno content type della response ricavato dal database, e poi inviare lo stream sulla response facendo uso di un buffer di output.

```
// recupero il file
stream = bean.downloadfile(idf);

// recupero il suo nome
filename = bean.getNomefile(idf);

// recupero il content type
String ct = bean.getContenttype(idf);

String cd = "attachment; filename=\""+filename+"\"";
            System.out.println("contentdisp :" + cd);
```

```
// setto la response sull'outputstream
ServletOutputStream  out = response.getOutputStream();

response.setContentType(ct);

// setto l'intestazione indicando il nome del file
if(cd!=null) {

            response.setHeader("Content-Disposition",cd);
        }

// E ora si invia lo stream di byte al client
bis = new BufferedInputStream(stream);
bos = new BufferedOutputStream(out);
byte[] buff = new byte[2048];
int bytesRead;

while(-1 != (bytesRead = bis.read(buff, 0, buff.length))) {
   bos.write(buff, 0, bytesRead);
}

out.flush();
```

L'upload di un file viene gestito attraverso la servlet continuaesegui che si occupa di fare il parsing della request di tipo multipart/form-data facendo uso delle API di O'Reilly.

Il file di cui si desidera fare l'upload viene temporaneamente salvato nella directory opportuna; se esiste già in tale directory un file con lo stesso nome, allora viene chiesto all'utente se sostituirlo o meno, altrimenti tutto viene lasciato così e si procede al salvataggio dei dati sul database. Per poter fare il salvataggio del file anche se ha un nome uguale ad un file già presente, si è utilizzata una procedura che costruisce una stringa con un nome di file che non esiste e salva temporaneamente il file con tale nome; in seguito alla decisione dell'utente la procedura effettua la cancellazione del file temporaneo o il suo rename con il nome corretto.

La servlet continuaesegui gestisce anche l'upload del file della foto del docente il cui salvataggio, in questo caso, non necessita di alcuna conferma in caso di omonimia.

Infine si osserva che per sapere se l'upload riguarda un file di materiale didattico o spazio libeo o foto, è sufficiente testare una variabile "stato".

```
// se bisogna eseguire l'aggiunta di un file occorre
beccare i parametri dal multipart form data

if( httpMethod.equals("POST") && !(reqContentType == null)
&& reqContentType.startsWith("multipart/form-data") ){

  int flag = 0;  /* flag per sapere cosa fare del file dopo
aver fatto il parsing della multpart request*/

String s = "nomenuovo";
/* stringa che indica un nome di file non esistente nel
path per salvataggio temporaneo*/

MultipartParser mp = new MultipartParser(request,
maxrequestdim);

Part reqpart = mp.readNextPart();

while(reqpart!=null) {

  if(reqpart.isParam()) {

      ParamPart p = (ParamPart)reqpart;

      if(p.getName().equals("desc")) {
 // recupera la descrizione
          desc = p.getStringValue();
      }

      if(p.getName().equals("id")) {
 // recupera l'id del titolo a cui aggiungere il file
          id = p.getStringValue();
```

```
      }
                              }
   if(reqpart.isFile()) {

      fp = (FilePart)reqpart; // recupero il file
      nome = fp.getFileName();
      contenttype = fp.getContentType();


   if(!stato.equals("insdoc")) {

      // eseguo l'upload di un file
      /* prima di salvare il file devo verificare che non
esista gia' un file con quel nome */

      if(bean.FileEsistente(nome)) { /* se il file esiste giÓ
chiedere conferma prima di sostituire  */

      flag = 1;  /* condizione che indica il salvataggio
soggetto a conferma */

      // salva il file con un nome che certamente non
esiste gia'
      s = nome;
      File f = new File(path + File.separator + s);

      while(f.exists()){
            s = "nuovo"+s;
                  f = new File(path + File.separator + s);

      }

      /* imposta il nome assoluto del file temporaneo*/

      fullname = path + File.separator + s;

      Msg nomesenzaspazi = new Msg(nome," ");

      // imposta la pagina cui andare a quella di richiesta
della conferma
```

```
      nextpage =
"/md/matdid/conferma.jsp?st="+stato+"&nome=" +
nomesenzaspazi.cambiamessaggio("+");
       }

       else { /* il file va salvato*/

            flag = 2;  // condizione che indica il
salvataggio immediato del file
            fullname = path + File.separator + nome;

       }

    } else  { // eseguo l'upload della foto

             fullname = pi.getFotoPath() + File.separator +
nome;

             // il flag resta a 0


            }

    // salvo il file nel file system
       is = fp.getInputStream();

      salvato = salvafile(is, fullname);
                                  }
       reqpart = mp.readNextPart();
}

       if(!salvato){  /* se il file non e' stato salvato
correttamente nel file system il lancio dell'eccezione
impedisce
il proseguimento dell'azione che farebbe il salvataggio sul
database e commit resta false */

          throw new Exception("Non e' stato possibile
salvare il file correttamente");

       }
```

```
   /* a questo punto ho recuperato tutti i parametri e
salvato il file */

     // in base al valore del flag stabilisco come agire sul
file

     switch(flag) {

        case 0 : // salvataggio immediato della foto

                 pi.nuovafoto(nome, contenttype);

                 commit = true;
                 break;


        case 1 : // salvataggio momentaneo del file
                 /* i valori vengono salvati
temporaneamente sul bean, senza inserire nulla nel
database*/

                 bean.salvatemporaneo(id, s , nome,
contenttype, desc );
                 commit = true;
                 break;

                 case 2 : // salvataggio immediato del file

                          bean.nuovoFile(id, nome,
contenttype, desc, true );
                          commit = true;

                          break;

     }  // fine case
```

I fogli di stile

Viene infine presentato un framento di un foglio di stile al fine di sottolinearne le particolarità. Si riporta a titolo di esempio il foglio di stile che consente di modificare la parte di spazio libero. Accanto alle diciture di titoli, files, links e note si trovano dunque i vari bottoni di modifica.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">

  <!-- contiene 1 se la pagina e' pubblicata -->
  <xsl:variable name = "pub">
                <xsl:value-of select="pag/@pubblicata"/>
  </xsl:variable>
<table width="559" border="0" cellpadding="5"> <!--
tabtitoli 1 riga per ogni titolo, ciascuna con 1 colonna --
>  // tabella con il contenuto da visualizzare
  <tr>

    <xsl:for-each select="pag/titolo">

      <xsl:sort select="@ordine" data-type="number"/>

      <!-- info per ciascun titolo -->
      <xsl:variable name = "idtitolo">
                <xsl:value-of select="@id"/>
      </xsl:variable>

      <!-- ... -->
      <!--  ogni titolo þ accompagnato dai bottoni di
modifica -->
                <!--  questa þ una parte della tabella che
riporta tali bottoni -->
<table border="0" cellspacing="6" cellpadding="2">  <!--
bottoni dei titoli 1 righe 5 colonne (una per ciascun
bottone) -->

                                        <tr>
```

```
                                                    <td
align="center"> <!-- bottone cancella titolo -->

                                                        <a
href="/md/esegui?az=c&amp;st=spldoc&amp;tipo=t&amp;id
={$idtitolo}">
                    <img src="/md/img/cancella.gif"
alt="cancella il titolo e il suo contenuto"
border="0"></img>

                                                        </a>

                                                    </td>
                                                <!--  .......  -->

                                                  </tr>
</table>

<!--  .......  -->

</xsl:for-each>

  </table>  <!-- fine tabella titoli -->

  <br></br>

 </xsl:if>

  </xsl:template>

</xsl:stylesheet>
```

Conclusioni e sviluppi futuri

Durante lo sviluppo di questa tesi sono state affrontate diverse problematiche di progettazione e implementazione di una applicazione Web.

Si è fatto uso della piattaforma J2EE, uno standard molto diffuso nel panorama del Web e destinato ad aumentare il proprio consumo da parte degli sviluppatori, grazie alle sue caratterisctiche. In particolare, avendo adottato un approccio EJB-centric, si sono potuti sfruttare i servizi messi a disposizione dal EJB-container, tra i quali connection pooling e la gestione container-menaged delle transazioni sono stati i servizi maggiormente utilizzati.

Nello sviluppo dei vari componenti sono stati considerati dei design patterns che la J2EE raccomanda, come l'uso degli helper objects, dei JavaBeans e custom tag e la quasi totale eliminazione degli scriptlet nelle JSP.

Grazie alla architettura J2EE e all'utilizzo di XML e dei fogli di stile si è ottenuto un elevato grado di separazione tra presentation logic e business logic, che rappresenta oggi uno degli obiettivi principali nello sviluppo di applicazioni Web.

Infine il modello component-based offerto dalla piattaforma J2EE ha favorito l'utilizzo di un approccio di progettazione object-oriented nel quale si è fatto uso del modello UML.

Uno sviluppo futuro dell'applicazione di sicuro interesse è quello di implementare un client stand-alone che si connetta via protocollo RMI-IIOP

direttamente all'EJB-tier, possibilità favorita dal fatto che si è utilizzato uno standard come XML per trasmettere le informazioni all'esterno dell'EJB-tier.

Appendice A

Manuale utente

In questa sezione viene presentato il manuale utente per l'utilizzo dell'applicazione. Esso è stato organizzato in modo da fornire al docente una chiara visione delle funzionalità messe a sua disposizione facendo una simulazione illustrata dei passaggi più significativi.

A.1 Funzionalità dell'utente generico

In questa sezione vengono prese in considerazione le funzionalità dell'utente generico, cioè di chiunque decida di collegarsi all'applicazione per visionare l'homepage di un docente.

A.1.1 Accesso all'applicazione

L'utente generico, connettendosi all'URL http://www.dbgroup.unimo.it:8063/md/hpdoc?iddoc=2 può liberamente consultare l'homepage del docente il cui id è quello specificato (iddoc=2).

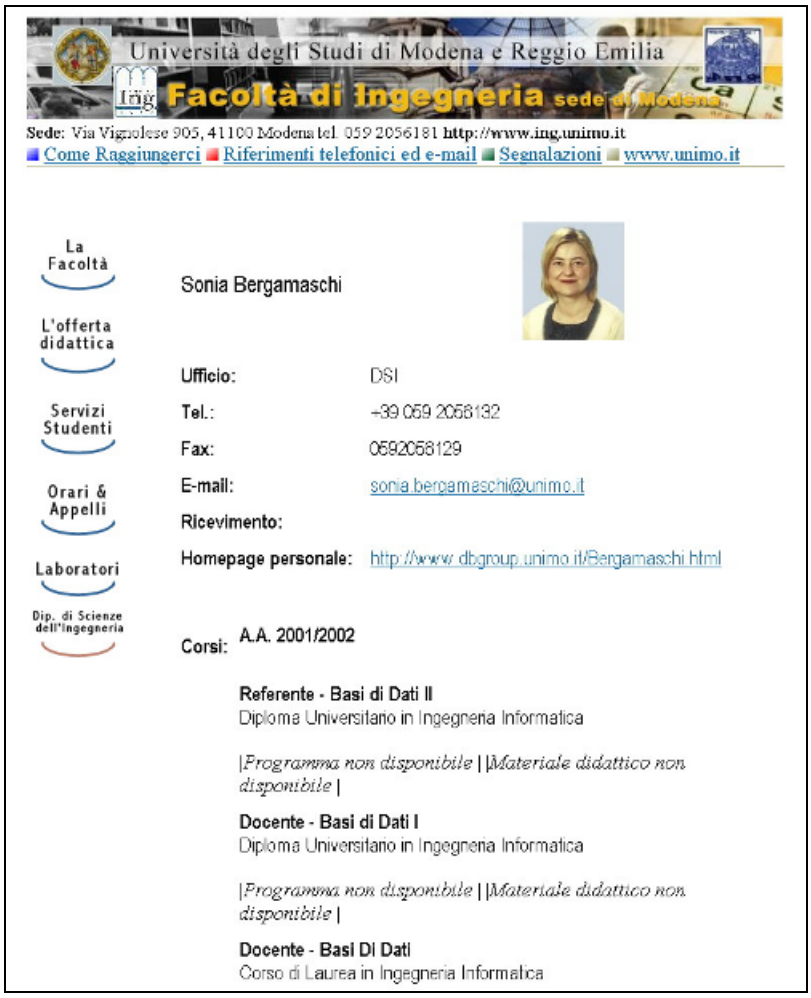


fig A.1 - Homepage docente vista dall'utente generico (parte alta) -

Nella figura A.1 è riportata la parte alta dell'homepage così come è vista dall'utente generico, nella quale si trovano i dati personali del docente e la sua foto (opzionale), e, a seguire, l'elenco degli insegnamenti da lui tenuti negli ultimi tre anni accademici. In questo elenco compaiono, per ogni insegnamento, due diciture in grassetto: la prima indica se il docente è effettivamente il docente dell'insegnamento o ne è solo il referente, la seconda riporta il nome dell'insegnamento; subito sotto compare, in caratteri normali, il nome del corso di cui l'insegnamento fa parte e più sotto ancora due diciture in corsivo corrispondenti a due links: il primo al programma dell'insegnamento e il secondo al materiale didattico. Ovviamente questi due ultimi

link potrebbero non essere ancora disponibili, nel qual caso non vi sarà alcun collegamento, ma semplicemente una dicitura che ne indica la non disponibilità.

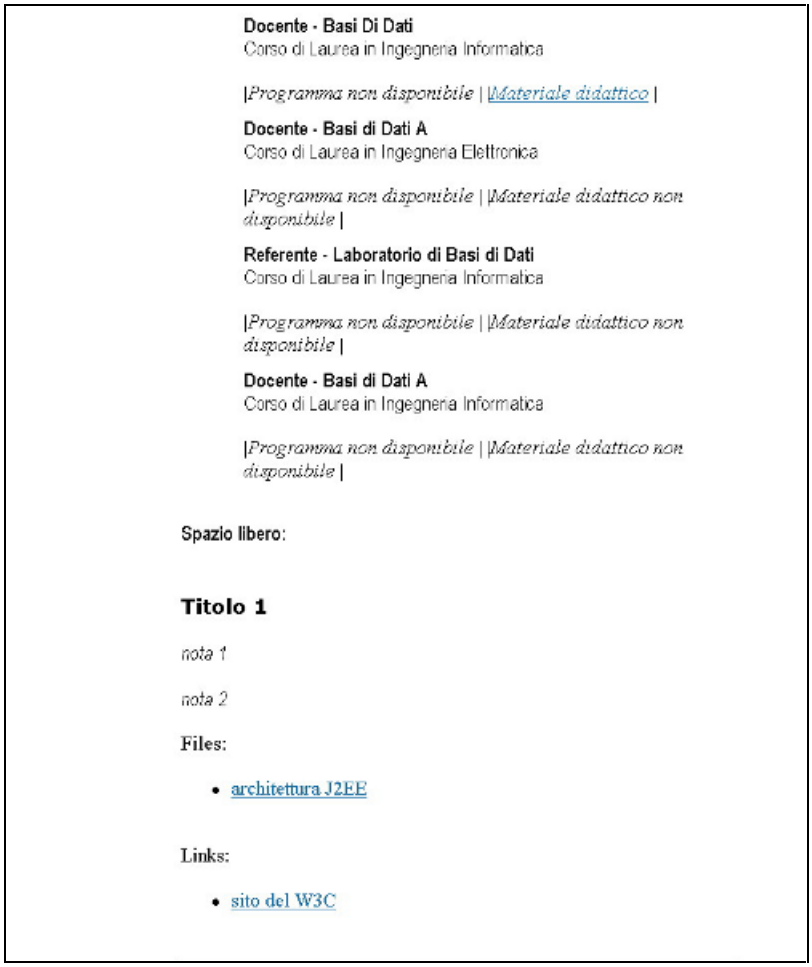


fig. A.2 - Homepage docente vista dall'utente generico (parte bassa) -

Nella figura A.2 viene riportata la parte bassa della pagina, nella quale si può notare, subito sotto all'ultimo insegnamento dell'elenco, la parte di spazio libero lasciata a discrezione del docente. Si noti la struttura di quest'ultima parte organizzata per argomenti, ciascuno dei quali ha un titolo, delle note (in corsivo), dei file scaricabili e dei link.

A.1.2 Altre funzionalità a disposizione dell'utente generico

A partire dalla homepage del docente, l'utente generico ha la possibilità di usufruire dei files e dei links che compaiono all'interno della parte di spazio libero.

Egli infatti può liberamente scaricare i files e connettersi ai link esterni che il docente ha deciso di pubblicare in questa parte della pagina.

E' importante sottolineare quindi che il materiale pubblicato in questa parte della pagina è liberamente accessibile da chiunque senza bisogno di alcun tipo di autorizzazione.

Naturalmente l'utente generico ha la possibilità di accedere anche a tutti gli altri link esterni all'applicazione, come quello al programma di un insegnamento, o gli altri links della fila verticale di bottoni sulla parte sinistra della pagina.

Un discorso a parte serve per il link al materiale didattico. Questo infatti è soggetto ad una restrizione che è lasciata a discrezione del docente, che può decidere di consentire l'accesso alla pagina a chiunque o solo ad utenti autorizzati.

In tutti i casi l'utente generico che clicca sul link al materiale didattico di un insegnamento si ritrova a dover inserire uno username e una password nell'apposito form riportato in figura A.3.

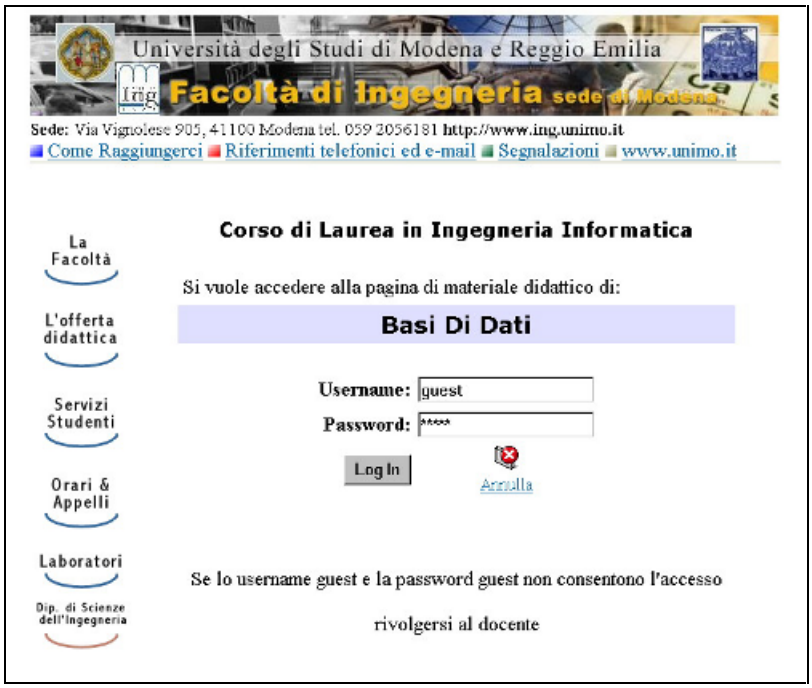


fig A.3 - Form di autenticazione per l'accesso al materiale didattico -

I campi di immissione di username e password compaiono già riempiti con i parametri di default stabiliti dal sistemista.

A questo punto l'utente generico può cliccare sul pulsante "Log In" per tentare l'accesso. Se il docente non ha provveduto a cambiare username e password, allora l'utente generico può accedere alla pagina di materiale didattico richiesta, acquistando automaticamente lo status di utente studente. Per la trattazione dettagliata delle funzionalità a disposizione dell'utente studente si veda oltre.

Se il docente ha provveduto a cambiare lo username e la password di default, l'utente generico non avrà accesso alla pagina di materiale didattico e vedrà visualizzato un messaggio di errore.

A.2 Funzionalità dell'utente studente

Come si è già detto l'utente che intende accedere alla pagina di materiale didattico di un insegnamento deve autenticarsi attraverso l'inserimento di uno username e una password nel form di login riportato in figura A.3.

Se il docente non ha specificato uno username e una password diversi da quelli di default l'utente generico può acquistare automaticamente lo status di utente studente e accedere alla pagina, altrimenti dovrà essere a conoscenza dei nuovi username e password settati per poter accedere.

A.2.1 Pagina di materiale didattico

L'utente studente autenticato ha accesso alla pagina di materiale didattico di un insegnamento. In figura A.4 viene riportata, a titolo di esempio, una pagina di prova relativa all'insegnamento di "Basi di Dati" del corso di laurea in Ingegneria Informatica.

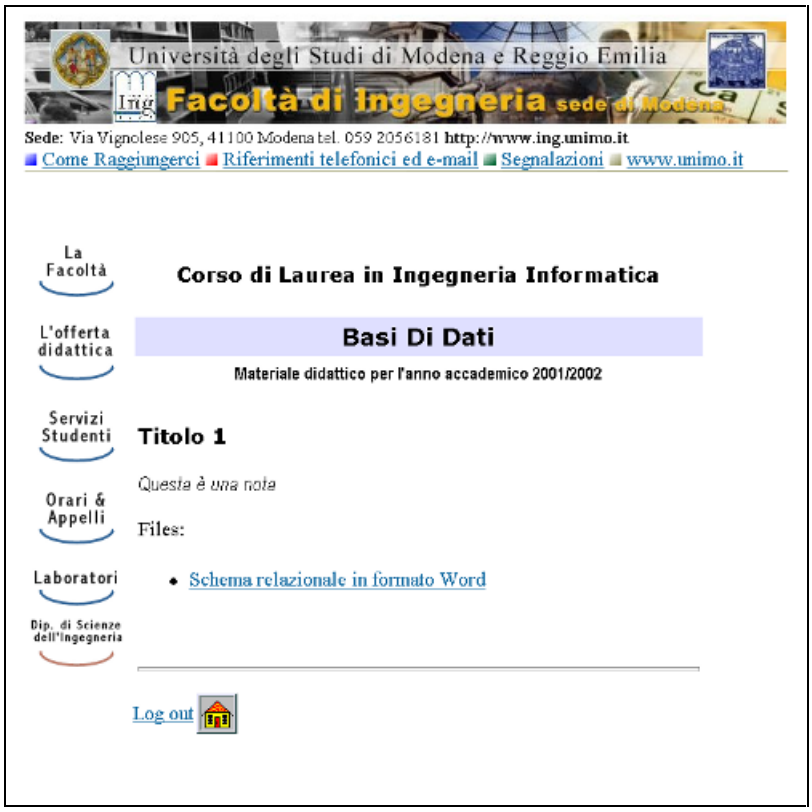


fig A.4 - Pagina di materiale didattico vista dall'utente studente -

Come si può notare la pagina è strutturata in modo del tutto analogo alla parte di spazio libero dell'homepage del docente, cioè è divisa per argomenti ciascuno con un titolo, delle note (in corsivo), dei file e dei link.

L'utente studente ha quindi la facoltà di scaricare i files e collegarsi ai siti pubblicati dal docente in questa pagina.

In fondo alla pagina si trovano due bottoni: uno con la dicitura "Log out" che consente di ritornare alla homepage senza mantenere lo status di utente studente, e l'altro con il simbolo della "casina" che consente di ritornare alla homepage mantenendo il proprio status di utente studente con lo username e la password immessi in precedenza.

Questa particolare differenza può essere notata controllando la parte bassa della homepage una volta ritornati. Infatti se si ritorna cliccando su "Log out" l'aspetto

della parte bassa è identico a quello di figura A.2, mentre se si ritorna cliccando sulla "casina" l'aspetto è quello di figura A.5, in cui si può notare la dicitura "Sei attualmente loggato con username guest" seguito dal bottone di "Log out".

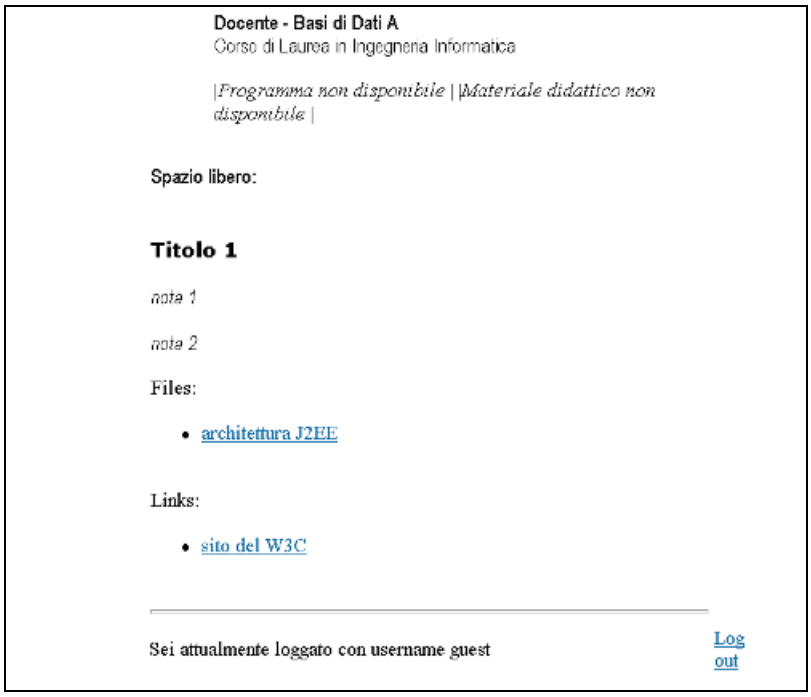


fig A.5 - Homepage vista dall'utente studente (parte bassa) -

E' importante sottolineare che mantenere lo status di studente è vincolante sui parametri di autenticazione, nel senso che, se si tenta la connessione ad un'altra pagina di materiale didattico il cui username e password sono diversi da quelli immessi precedentemente, ci si imbatte in un errore di autenticazione e l'accesso viene negato.

E' quindi opportuno ritornare alla homepage sempre col pulsante di "Log out" se si ha l'intenzione di cambiare pagina di materiale didattico. Il pulsante "casetta" può essere utile nel caso lo studente voglia consultare più pagine di materiale didattico differenti dello stesso docente, e sia già a conoscenza del fatto che queste richiedono tutte gli stessi username e password, evitando così di doverli reinserire di nuovo tutte le volte.

A.3 Funzionalità dell'utente docente

Si prendono ora in considerazione tutte le operazioni che un docente può effettuare per personalizzare la sua homepage e per pubblicare il materiale didattico dei suoi insegnamenti.

A.3.1 Accesso alla applicazione

L'utente docente che intende modificare la sua homepage deve connettersi all'URL http://www.dbgroup.unimo.it:8063/md/matdid/docentelogin.jsp ed inserire i propri username e password nel form di log in il cui aspetto è quello riportato in figura A.6.

fig A.6 - Form di autenticazione per l'utente docente -

Una volta utenticato, l'utente docente si ritrova automaticamente nella propria homepage, il cui aspetto è simile a quello visto dall'utente generico, ma con l'aggiunta di alcuni bottoni in più, come si può vedere dalla figura A.7.

fig A.7 - Homepage vista dall'utente docente (parte alta) -

I primi due bottoni disponibili si trovano subito sotto la parte di dati personali del docente, e sono : il bottone "cambia foto" e il bottone "modifica parte di spazio libero". Il primo consente di cambiare la propria foto che si vuole far comparire nell'homepage attraverso un opportuno form di immissione il cui aspetto è riportato

in fig A.8, mentre il secondo consente di accedere ad una pagina in cui è possibile modificare la parte di spazio libero dell'homepage.

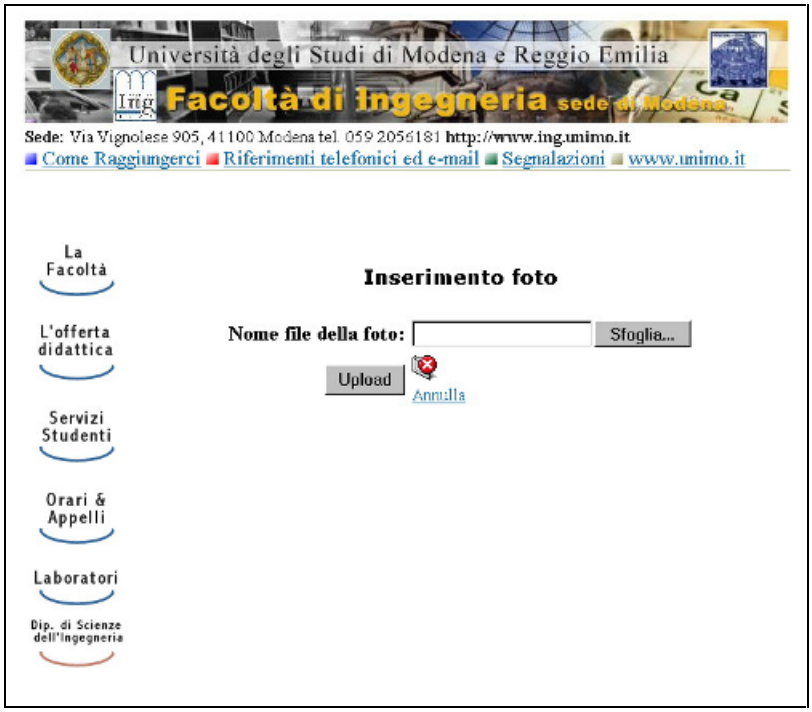


fig A.8 - Pagina di inserimento della nuova foto -

A.3.2 Modifica della parte di spazio libero

Nella figura A.9 è riportata la pagina di modifica della parte di spazio libero. Come si vede la pagina riporta, accanto a ciascuna voce, numerosi bottoni che consentono di effettuare operazioni differenti.

fig A.9 - Pagina di modifica parte di spazio libero -

Subito sotto al nome del docente a cui appartiene la pagina in modifica, è riportata una dicitura che indica se la parte di spazio libero della homepage è attualmente pubblicata o meno, cioè se è attualmente visibile dall'utente generico. Nell'esempio di figura A.9 la dicitura indica che la pagina è visibile all'utente generico che la vedrà come riportato in figura A.2.

Accanto a questa dicitura si trova un bottone che consente di alternare la situazione da "pubblicata" a "non pubblicata". Nell'esempio, essendo la parte di spazio libero già visibile, il bottone è quello che inibisce la pubblicazione.

Questa funzione è molto importante perché consente al docente di rendere pubblica la parte di spazio libero solo quando lo ritiene opportuno. Prima di iniziare

delle operazioni di modifica sarebbe opportuno inibire la pubblicazione per poi riattivarla solo quando tali operazioni sono terminate.

Il bottone successivo che si incontra scendendo nella pagina è quello “aggiungi titolo” che consente di inserire un nuovo argomento specificandone il titolo. Cliccandolo si passa ad un form di inserimento della descrizione del nuovo titolo che è riportato nella figura A.10.

fig A.10 - Form di inserimento di un nuovo titolo -

Se viene confermato l’inserimento del nuovo titolo, questo comparirà nella pagina di modifica immediatamente sotto all’ultimo titolo presente. La fig A.11 mostra un esempio di inserimento di un nuovo titolo con descrizione “Titolo 2”.

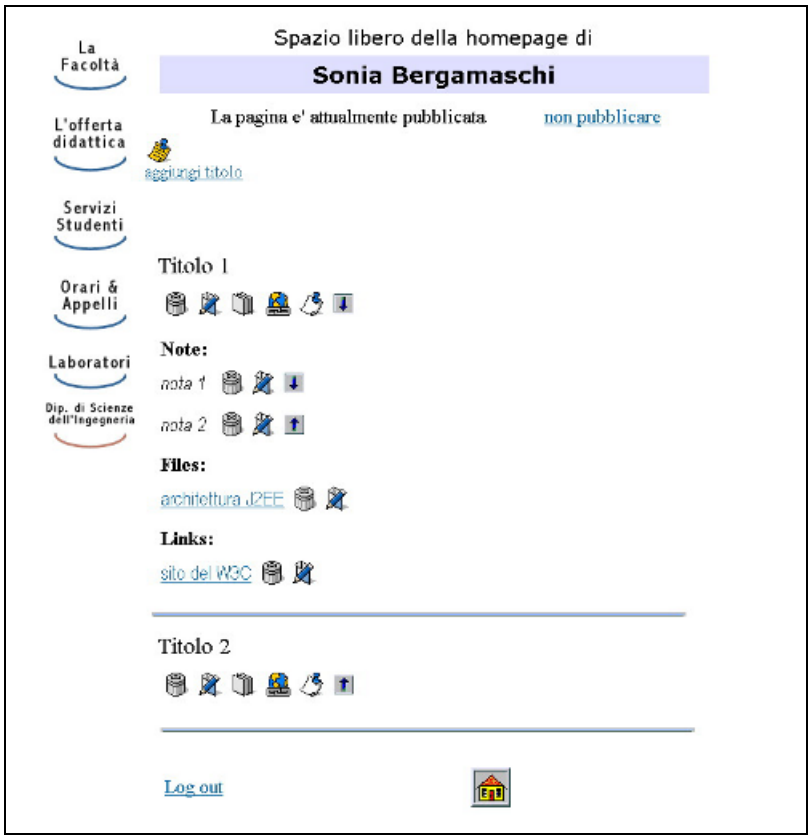


fig A.11 - Pagina di modifica con il nuovo titolo appena aggiunto -

Per quanto riguarda le funzioni degli altri bottoni della pagina, dei quali si riporta la legenda in figura A.12 , questi riguardano le varie operazioni di aggiunta, modifica, cancellazione e spostamento che verranno illustrate in seguito in modo dettagliato con degli esempi.

SIMBOLO	DESCRIZIONE
	Cancella l'elemento
	Modifica l'elemento
	Aggiungi un nuovo file
	Aggiungi un nuovo link
	Aggiungi una nuova nota
	Sposta l'elemento verso l'alto di una posizione
	Sposta l'elemento verso il vasso di una posizione

fig A.12 - Legenda delle icone -

Aggiunte di elementi all'interno di un titolo

Le aggiunte di elementi all'interno di un titolo si ottengono cliccando sul relativo bottone posto sotto alla descrizione del titolo stesso.

Ad esempio se si vuole aggiungere una nota al nuovo titolo appena creato basterà cliccare sul bottone corrispondente riportato sotto a "Titolo 2" e apparirà il form di figura A.13.

fig A.13 - Form di aggiunta di una nuova nota -

Cliccando sul bottone “Inserisci” la nota verrà aggiunta sotto all’ultima nota presente nel “Titolo 2”, così come mostrato in figura A.14.

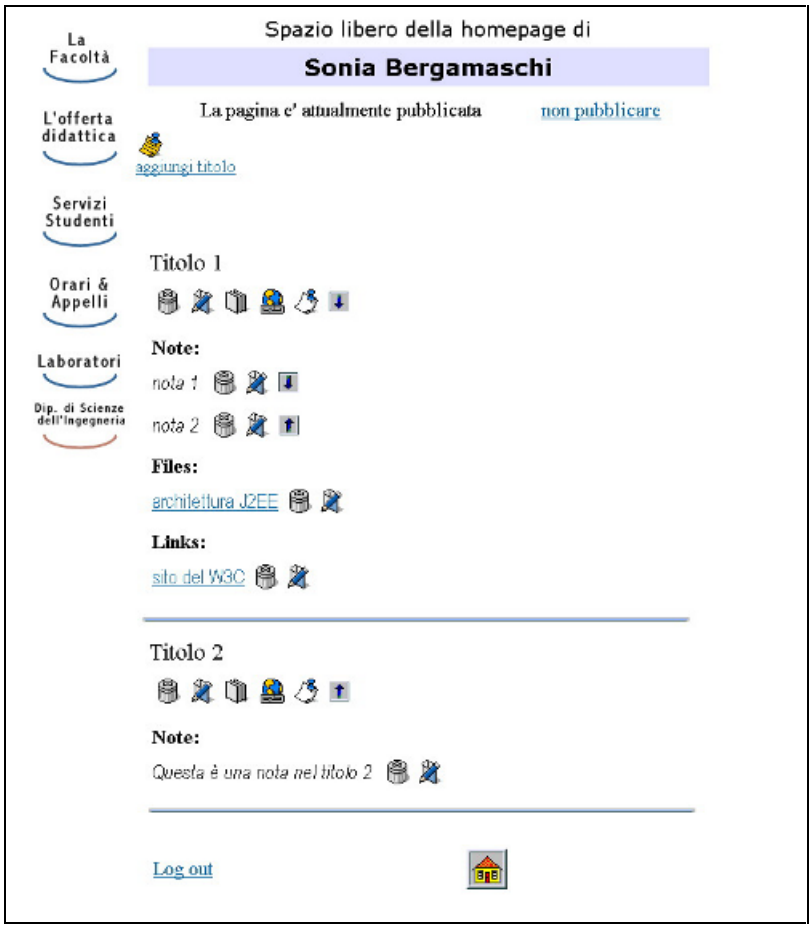


fig A.14 - Pagina di modifica con la nuova nota inserita -

L'aggiunta degli altri elementi è del tutto analoga. In figura A.15 e A.16 sono mostrati i form di inserimento rispoettivamente di un link e di un file. E' importante ricordare che tutti i campi di immissione dei form di aggiunta sono obbligatori.

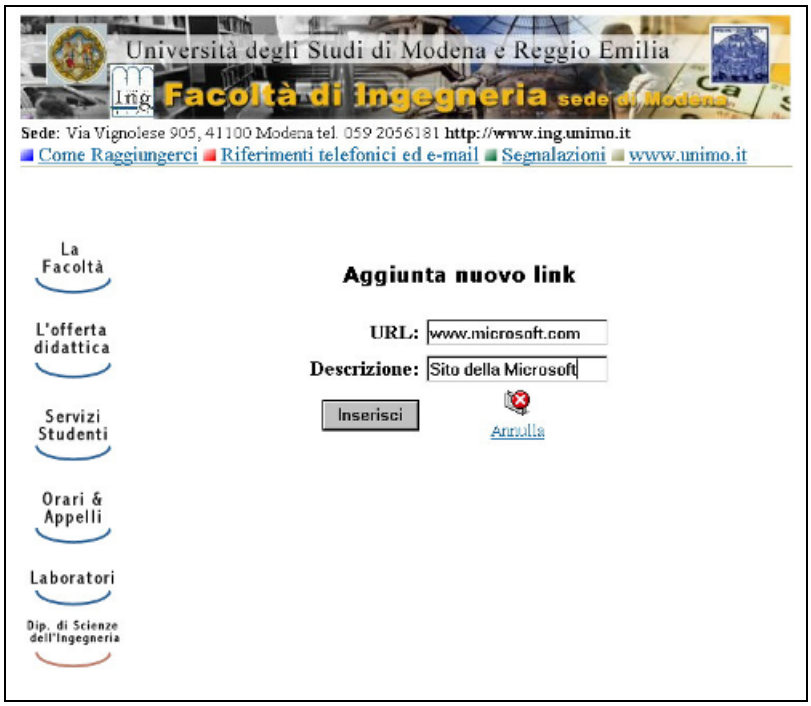


fig A.15 - Form di aggiunta di un nuovo link -

fig A.16 - Form di aggiunta di un nuovo file -

Il risultato delle operazioni fin qui fatte, nella fattispecie dell'esempio illustrato, sono riportate in figura A.17.

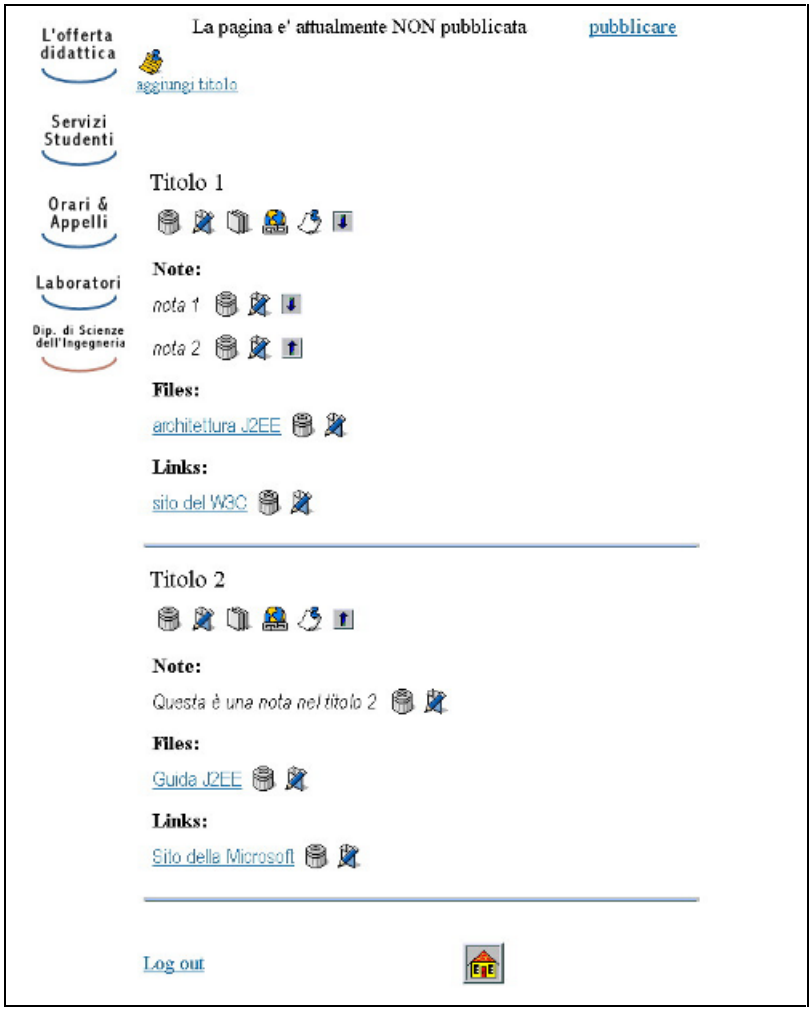


fig A.17 - Pagina di modifica con le aggiunte fatte -

Nel caso di aggiunta di un file il cui nome esiste già all'interno della directory assegnata dal sistemista, compare un messaggio di notifica, riportato in fig A.18, che chiede al docente quale comportamento deve essere seguito : sovrascrivere il file oppure annullare l'operazione.

fig A.18 - Form di notifica di file già esistente -

Cancellazione di elementi

Se il docente desidera cancellare un elemento può farlo cliccando sul bottone "cestino" posto a fianco a ciascun elemento all'interno di un titolo, o sotto alla descrizione del titolo. Quest'ultimo consente la cancellazione di un intero argomento con tutto il suo contenuto.

Spostamento di elementi

Le aggiunte di nuovi elementi avvengono sempre in una posizione che è quella immediatamente sotto l'ultimo elemento presente dello stesso tipo.

Se il docente desidera mutare l'ordine di comparizione di un qualsiasi elemento all'interno della pagina può farlo attraverso i bottoni frecciati associati (che compaiono a fianco dell'elemento stesso solo se vi sono più elementi dello stesso tipo all'interno di un titolo, oppure sotto al titolo se vi sono più titoli).

Ad esempio se il docente vuole invertire l'ordine di comparizione dei titoli, con il relativo contenuto, può cliccare indifferentemente sul bottone "freccia su" del "Titolo 2" o sul bottone "freccia giù" del "Titolo 1". Il risultato è quello riportato in figura A.19.

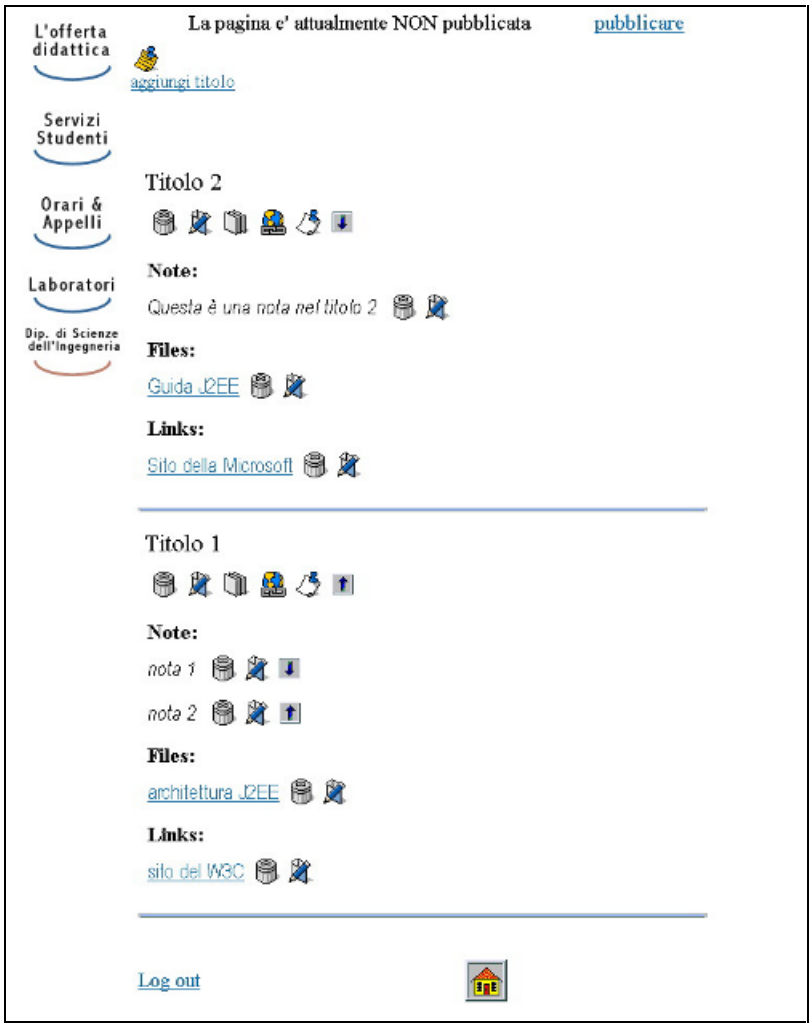


fig A.19 - Pagina delle modifiche dopo lo scambio dei titoli -

Modifiche ad elementi

Ognuno degli elementi della pagina (titolo, nota, file o link) può essere modificato cliccando sul corrispondente bottone di modifica.

Ad esempio se il docente vuole modificare la descrizione del file “Guida J2EE” del “Titolo 2” deve cliccare sul pulsante modifica posto accanto alla descrizione del file, e inserire il nuovo valore nel form di immissione, come quello riportato in figura A.20.

fig A.20 - Form di modifica della descrizione di un file -

Le modifiche degli altri elementi sono analoghe, a parte quello dei link il cui form di modifica presenta due campi di immissione, uno per l’URL e uno per la descrizione del link. Ad esempio se il docente vuole modificare il link “Sito del W3C” e clicca sul bottone di modifica corrispondente, si ritrova il form riportato in figura A.21. Inserendo la nuova descrizione lasciando vuoto l’URL si conferma implicitamente quest’ultimo. E’ bene ricordare infatti che nei form di modifica nessun campo di immissione è obbligatorio, poichè se un campo viene lasciato vuoto si intende riconfermato il valore precedente.

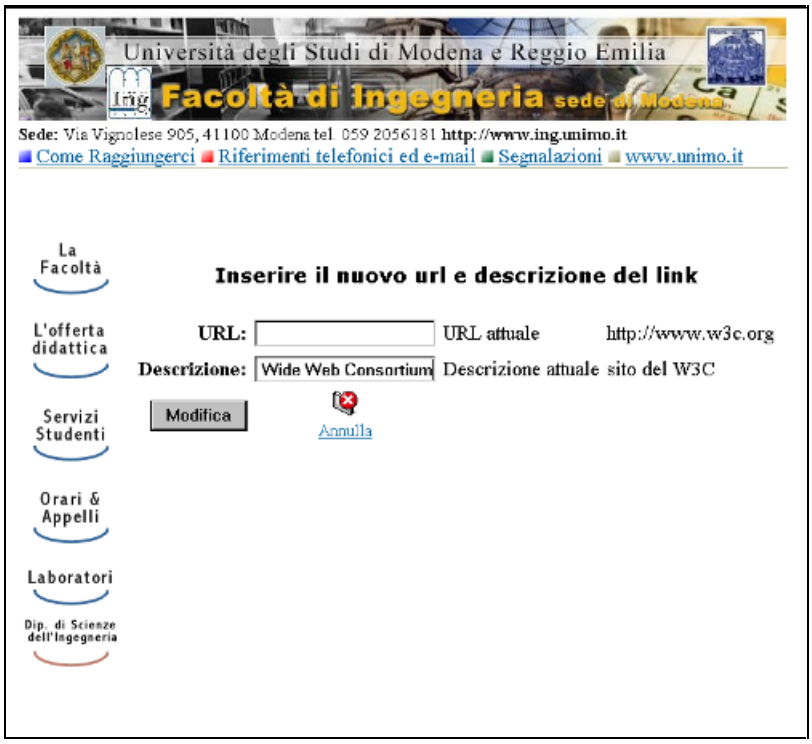


fig A.21 - Form di modifica di un link -

Ritornando alla pagina di modifica si osservano le modifiche appena apportate come in fig A.22.

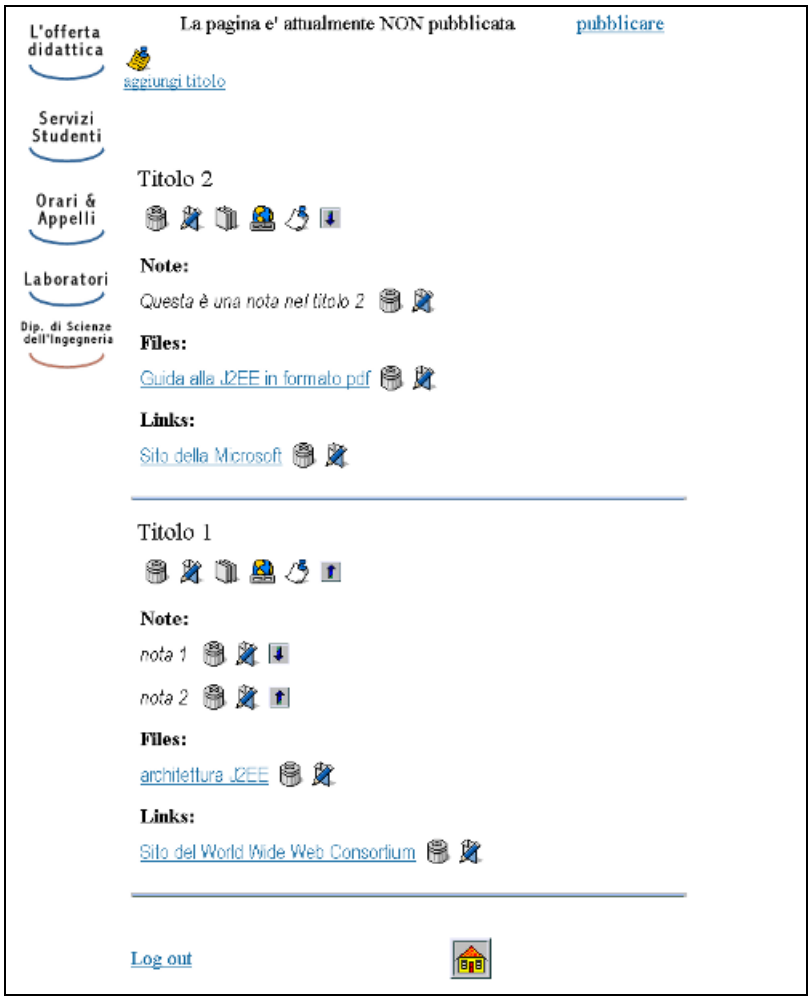


fig A.22 - Pagina di modifica con le modifiche apportate -

Ripubblicazione

Una volta finiti gli aggiornamenti alla parte di spazio libero, il docente può decidere di renderli pubblici cliccando sul pulsante “pubblicare”. Il risultato sarà che l’utente generico, collegandosi alla homepage del docente, vedrà la nuova versione della parte di spazio libero, come in figura A.23.

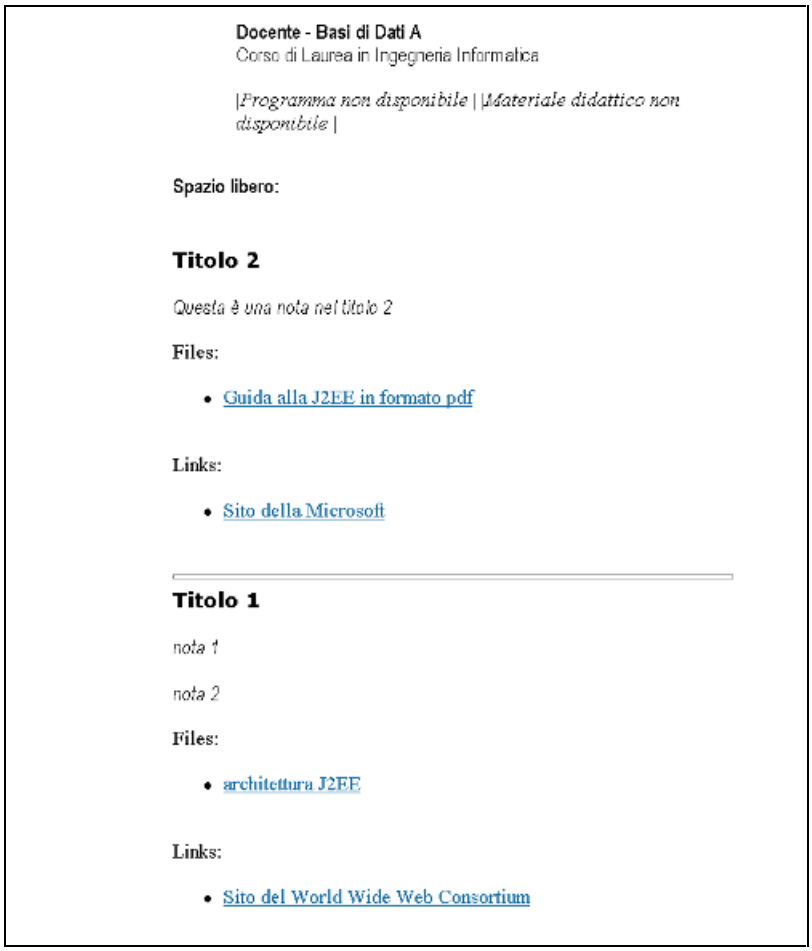


fig A.23 - Homepage vista dall'ut. generico dopo la ripubblicazione (parte bassa) -

A.3.3 Modifiche della pagina di materiale didattico di un insegnamento

Fin qui si è parlato delle funzionalità di aggiornamento della parte di spazio libero dell'homepage del docente, ma le stesse funzionalità sono applicabili alle pagine di materiale didattico di ciascun insegnamento tenuto.

Se il docente ad esempio clicca sul bottone "modifica materiale didattico" dell'insegnamento di "Basi di Dati" del corso di laurea in Ingegneria Informatica, la pagina che compare, riportata in figura A.24, è strutturata in modo del tutto analogo a quella di modifica dello spazio libero.

fig A.24 - Pagina di modifica del materiale didattico di un insegnamento -

E' dunque possibile fare operazioni di aggiunta, modifica, cancellazione, spostamento e pubblicazione/Non-pubblicazione esattamente come già illustrato per la parte di spazio libero.

L'unica novità è la possibilità che viene data in questa sede di cambiare la password, ed è questo il primo bottone che si trova sotto il nome dell'insegnamento cui la pagina si riferisce. A sinistra del bottone compare una dicitura che indica quali sono gli username e la password attuali (se sono quelli di default o se sono quelli immessi dal docente).

Cliccando il pulsante il docente si ritrova a dover compilare un form con l'aspetto riportato in figura A.25 , nel quale viene chiesta l'immissione di un nuovo username, una nuova password e la conferma della nuova password.

fig A.25 - Form di immissione nuovi username e password -

L'aspetto della pagina di modifica del materiale didattico dopo questo cambiamento di password risulta quello riportato in figura A.26.

fig A.26 - Pagina di modifica materiale didattico dopo aver cambiato password -

Appendice B

Upload di file via HTTP

In questa appendice si descrive sinteticamente l'invio di file tramite il protocollo HTTP.

Per una trattazione più dettagliata, che esula dagli scopi di questa tesi, si rimanda alle specifiche HTTP [HTTP] e HTML [HTML] .

B.1 Form HTML

I form HTML vengono utilizzati per consentire al Web-client l'invio di dati mediante una request HTTP.

Per inserire un form all'interno di una pagina Web si utilizza il tag HTML `<form>`. L'attributo `action` specifica l'URI di una risorsa Web al quale deve essere inviato il contenuto del form per essere elaborato. L'attributo `method` specifica quale metodo HTTP deve essere usato per inviare tale contenuto. Con il metodo GET l'insieme dei dati del form viene aggiunto all'URI indicato dall'attributo action con il carattere "?" come separatore. Con il metodo POST i dati vengono inclusi nel body della request.

L'utente interagisce con i form mediante i cosiddetti controlli. Questi consentono l'invio di semplici coppie nome=valore oppure di dati più complessi.

I tag `<input>` inseriti nel body di un tag `<form>` indicano quali controlli si vogliono includere e gli attributi `type`, `id` e `value` definiscono rispettivamente il tipo, il nome e il valore iniziale del controllo.

Qui di seguito viene riportato un esempio di form HTML per l'invio di due stringhe di testo con il metodo POST.

```
<FORM action="http://somesite.com/prog/adduser"
method="post">
      <INPUT type="text" id="nome"><BR>
      <INPUT type="text" id="cognome"><BR>
      <INPUT type="submit" value="Invia">
      <INPUT type="reset"> </BR>
</FORM>
```

- Esempio di un form HTML per l'invio di stringhe di testo -

B.2 Invio dei form

L'attributo `enctype` del tag `<form>` specifica il tipo di contenuto della request HTTP che viene inviata al server.

Il tipo di contenuto *application/x-www-form-urlencoded* , che è il valore di default dell'attributo `enctype`, viene utilizzato per l'invio di quantità limitate di dati e in genere comprendenti soli caratteri ASCII.

Il tipo di contenuto *multipart/form-data* è in genere utilizzato per inviare grosse quantità di dati, inclusi file, e utilizza un tipo di codifica in grado di inviare tutti i caratteri previsti dall'*ISO-10646* [UCS] . L'invio di un form con questo tipo di enctype segue le regole di tutti i flussi di dati MIME in più parti, come è stato delineato in [RFC2045] . Ogni parte, separata dalla successiva da una sequenza di caratteri detta *boundary* , corrisponde ad un controllo a buon esito ed è costituita da una intestazione e da un body. Nell'intestazione devono comparire un header field

denominato Content-Disposition il cui valore è “form-data”, e un header field Content-Type il cui valore è una sigla standard che descrive il tipo di contenuto del body. L’elenco di queste sigle standard è riportato nel registro ufficiale IANA [IANA] . Il content type dell’intero flusso di dati ha valore “multipart/form-data” e deve essere riportato nell’intestazione assieme al boundary.

Si riporta qui di seguito un esempio di form HTML per l’invio di un flusso multipart che contiene due parti : una stringa di testo e un file.

```
<FORM action="http://server.dom/cgi/handle"
enctype="multipart/form-data" method="post">
      <P> Come ti chiami?
      <INPUT type="text" name="submit-name">
      <BR> Quali file vuoi inviare?
      <INPUT type="file" name="files"><BR>
      <INPUT type="submit" value="Invia">
      <INPUT type="reset">
</FORM>
```

- Esempio di form HTML per l’upload di un file -

Se l’utente scrivesse “Andrea” nel campo di immissione di tipo testo e inviasse il file “prova.txt” nel controllo di tipo file, il flusso multipart inviato sarebbe, secondo le specifiche MIME, del tipo seguente:

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"
Andrea
--AaB03x
Content-Disposition: form-data; name="files";
filename="prova.txt" Content-Type: text/plain
... contenuto di prova.txt ...
--AaB03x
```

- Flusso MIME del form precedente -

B.3 I Character Set

Come è noto ogni carattere per poter essere memorizzato e utilizzato da un computer va convertito in un numero intero, che può essere rappresentato mediante codifica binaria.

Per *Character Set* (o *charset*) si intende un criterio che associa univocamente un set di byte ad un set di caratteri. Esistono numerosi charset diversi a seconda delle particolari esigenze di linguaggio che supportano. Il più semplice e noto charset è l'*ANSI X3.4-1986* , conosciuto anche come *ASCII* (*7-Bit American Standard Code for Information Interchange*), che utilizza i 7 bit meno significativi del byte per codificare fino a 128 caratteri differenti. Esso comprende le lettere dell'alfabeto utilizzate per la lingua inglese, i numeri e alcuni altri caratteri speciali di uso comune. Ad esempio la lettera "J" viene codificata con l'intero decimale 74 che tradotto in binario corrisponde a `1001010` . Il bit più significativo è utilizzato come bit di parità.

L'ASCII è un esempio di SBCS (Single Byte Character Set) che mappa un byte per carattere, ma ve ne sono numerosi altri, la maggior parte dei quali sono sovrainsiemi dell'ASCII che utilizzano anche il bit più significativo per codificare fino a 256 caratteri. Tra questi i più noti sono gli ISO (International Organization for Standardization) character sets e gli ANSI (American National Standards Institute) character sets. I primi sono utilizzati dai sistemi operativi Mac e il più diffuso è senz'altro l'*ISO-8859-1*, conosciuto anche come *Latin-1* che è in grado di codificare quasi tutti i caratteri degli alfabeti delle lingue occidentali. I secondi sono utilizzati dai sistemi operativi Windows 3.x/9x e quello che merita più attenzione è il *ANSI-1252*, conosciuto anche come *Windows-1252*, che mappa gli stessi caratteri del Latin-1, con l'aggiunta di altri caratteri particolari che occupano le posizioni dal 128 al 159 che nel Latin-1 non sono definite (ad esempio il carattere corrispondente al 128 è definito come simbolo dell'euro).

Quando vengono inviate stringhe di caratteri o file di testo mediante HTTP ogni browser utilizza il proprio charset per convertire i caratteri in byte. Se il server di

destinazione utilizza un charset differente si possono creare dei problemi di conversione dei byte ricevuti in caratteri.

Questo problema potrebbe essere superato se si potesse far uso di un unico charset che mappasse tutti i possibili caratteri degli alfabeti mondiali. Un tentativo in questo senso è stato compiuto con l'*Unicode* [UC] , un DBCS (Double Byte Character Set) che mappa ogni carattere su due bytes, il meno significativo dei quali corrisponde al charset Latin-1, mentre il più significativo ricopre tutti gli altri caratteri, compresi i caratteri delle lingue asiatiche, arrivando così ad un massimo di 65.536 caratteri mappabili. Sfortunatamente non tutti i sistemi operativi (ad esempio Windows 3.x/9x) e non tutti i DBMS supportano l'Unicode, per cui appare ancora lontana la scomparsa di tutti i charset a favore del solo Unicode.

Per lo scopo di questa tesi e l'ambito dell'applicazione realizzata, si è assunto che tutti i dati inviati e ricevuti dal server via HTTP facessero uso di una unica codifica compatibile con la Latin-1.

Per quanto riguarda la memorizzazione dei dati su SQL Server, essendo Latin-1 il charset di default, non è stata apportata nessuna modifica di configurazione e tutti i campi testuali delle tabelle sono stati definiti con il tipo *VARCHAR* che utilizza un mapping SBCS. Si tenga presente comunque che SQL Server dalla versione 7 in poi supporta la codifica Unicode attraverso il tipo *NVARCHAR* che utilizza un mapping DBCS.

Bibliografia

[NORM] Beneventano, Bergamaschi, Vincini, “PROGETTO DI BASI DI DATI RELAZIONALI”, Pitagora Editrice - Bologna

[J2EE] “Java 2 Enterprise Edition” documentazione disponibile al sito http://java.sun.com/j2ee/docs.html . J2EE SDK scaricabile al URL http://java.sun.com/j2ee/download.html. Un ottimo tutorial per sviluppatori di applicazioni J2EE è disponiile al URL http://java.sun.com/blueprints/index.html

[SUN] Sito ufficiale di Java della Sun Microsystem : http://java.sun.com.

[SPECEJB] Specifiche degli Enterprise Java Beans scaricabili al URL http://java.sun.com/products/ejb/docs.html

[TOMCAT] Sito di Tomcat : http://jakarta.apache.org/tomcat/index.html

[JBOSS] Sito ufficiale : http://jboss.org . Sito di SourceForge : http://sourceforge.net/

[JAKARTA] Sito ufficiale dl progetto Jakarta : http://jakarta.apache.org/

[UML] Martin Fowler. UML Distilled. Guida rapida allo Standard Object Modeling Language, Addison Wesley.

[MVC] “Model View Controller - J2EE Design patterns” documentazione al URL http://java.sun.com/blueprints/patterns/j2ee_patterns/model_view_controller/

[DAO] “Data Access Object - J2EE Design patterns” documentazione disponibile al URL
http://java.sun.com/blueprints/patterns/j2ee_patterns/data_access_object/

[VO] “Value Object - J2EE Design patterns” documentazione disponibile al URL http://java.sun.com/blueprints/patterns/j2ee_patterns/value_object/

[JAXP] “Java API for XML Processing” documentazione disponibile al URL http://java.sun.com/xml/jaxp/index.html

[DOM] “Document Object Model” standard definito dal World Wide Web Consortium. Documentazione disponibile al URL http://www.w3c.org/DOM/

[OREILLY] Package com.oreilly.servlet delle API di supporto per l’upload di files scaricabile al URL http://www.servlets.com/cos/index.html

[XSLTL] “XSL Taglib” scaricabile dal sito del Progetto Jakarta all’indirizzo http://jakarta.apache.org/taglibs/doc/xsl-doc/intro.html. La lista completa delle tag libraries del Progetto Jakarta è disponibile al URL http://jakarta.apache.org/taglibs/index.html

[HTTP] “HyperText Transfer Protocol” documentazione disponibile al URL http://www.w3c.org/Protocols/

[HTML] “HyperText Markup Language” documentazione disponibile al URL http://www.w3c.org/MarkUp/

[UCS] “Information Technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane”

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed e N. Borenstein, Novembre 1996.

Il file di testo del RFC2045 è disponibile al URL http://www.cis.ohio-state.edu/cs/Services/rfc/rfc-text/rfc2045.txt

[IANA] "Assigned Numbers", STD 2, RFC 1700, USC/ISI, J. Reynolds e J. Postel, Ottobre 1994. Disponibile in formato testo al URL http://www.cis.ohio-state.edu/cs/Services/rfc/rfc-text/rfc1700.txt

[UC] "The Unicode Standard: Version 2.0", The Unicode Consortium, Addison-Wesley Developers Press, 1996. Per maggiori informazioni, si consulti la home page dell'Unicode Consortium all'indirizzo http://www.unicode.org