UNIVERSITÀ DEGLI STUDI DI MODENA
E REGGIO EMILIA
Facoltà di Ingegneria – Sede di Modena

Corso di Laurea in Ingegneria Informatica

Progetto e sviluppo di un'applicazione Web-Database a 3 livelli per la gestione degli appelli d'esame

Relatore

Chiar.mo Prof.

Sonia Bergamaschi

Correlatore

Ing. Maurizio Vincini

Tesi di Laurea di

Andrea Malavasi

Anno Accademico 2001-2002

Ringraziamenti

Desidero ringraziare la Professoressa Sonia Bergamaschi per l'aiuto fornito durante la realizzazione della tesi e per la disponibilità dimostrata.

Un ringraziamento anche all'Ing. Maurizio Vincini e l'Ing. Francesco Guerra per la collaborazione e l'interesse dimostrato nello svolgimento di questa tesi di laurea.

Infine desidero ringraziare la mia famiglia ed i miei amici per il sostegno dato e per l'enorme pazienza dimostrata in questi sei mesi.

Sommario

Elenco figure

1 Specifiche dei requisiti

2 Enterprise Java Bean e Java Server Page

3 Analisi dell'applicazione web con UML

Appendice

CAPITOLO 1
Specifiche dei requisiti

1.1 Problematiche di progetto

I paragrafi che seguono affrontano alcuni aspetti di cui bisogna tenere conto nelle fasi iniziali del progetto. Questo tipo di approccio è fondamentale per evitare di dover rivoluzionare il progetto in fase di elaborazione. Infatti, risulta essere molto più efficace affrontare inizialmente tutte le problematiche di progetto nella loro totalità piuttosto che dover, di volta in volta, interrompere il lavoro per affrontare separatamente ciascuna parte dell'implementazione. Questo non vuol certo dire che non possano insorgere dei problemi durante lo sviluppo, si vuole soltanto affermare che il loro numero viene ridotto drasticamente.

1.2 Considerazioni generali

Come qualsiasi applicazione software anche le applicazioni web necessitano di abbondanti specifiche dei requisiti per essere modellate in modo efficiente. Le specifiche dei requisiti si dividono in specifiche funzionali e non funzionali. Le prime descrivono da parte del committente ciò che deve svolgere l'applicazione, le seconde riguardano che tipo di tecnologia sta alla base dell'implementazione dell'applicazione. L'applicazione che si desidera realizzare riguarda alcuni aspetti per la gestione degli appelli d'esame in ambito universitario. Le specifiche funzionali che verranno di seguito riportate riguardano in gran parte l'applicazione realizzata nel corso della tesi ed in parte documenti

precedentemente realizzati. Quanto appena detto è spiegato dal fatto che questa applicazione web verrà presto integrata nel sito della facoltà di ingegneria e quindi, non può prescindere dalle scelte di progetto già compiute.

Il portale web di facoltà rappresenta senza dubbio un applicazione web molto complessa, sia per la vastità del progetto, che per le peculiarità dell'ambiente universitario. Il sito web deve dunque essere concepito con una struttura molto flessibile ed aperta, in grado di adattarsi ad eventuali variazioni future.

Quello che è importante notare è che il sito della facoltà ed in particolar modo questa applicazione non devono soltanto soddisfare le esigenze di coloro che accedono al sito per ottenere informazioni ma anche di tutti coloro che vi accedono per mantenerle aggiornate. Per questo motivo l'accesso all'area di amministrazione del sito web, dovrà essere in grado di discriminare la tipologia dell'utente che ha avuto accesso. Ogni tipo di utente avrà quindi la possibilità di accedere esclusivamente alle informazioni di sua competenza.

1.3 Vincoli Progettuali

Nello sviluppare quest'applicazione si è fatto riferimento ad una nuova tecnologia che grazie alla sua notevole efficienza e versatilità ha raggiunto, in breve tempo, un elevato grado di diffusione e consenso.

1.3.1 J2EE (Java 2 Enterprise Edition)

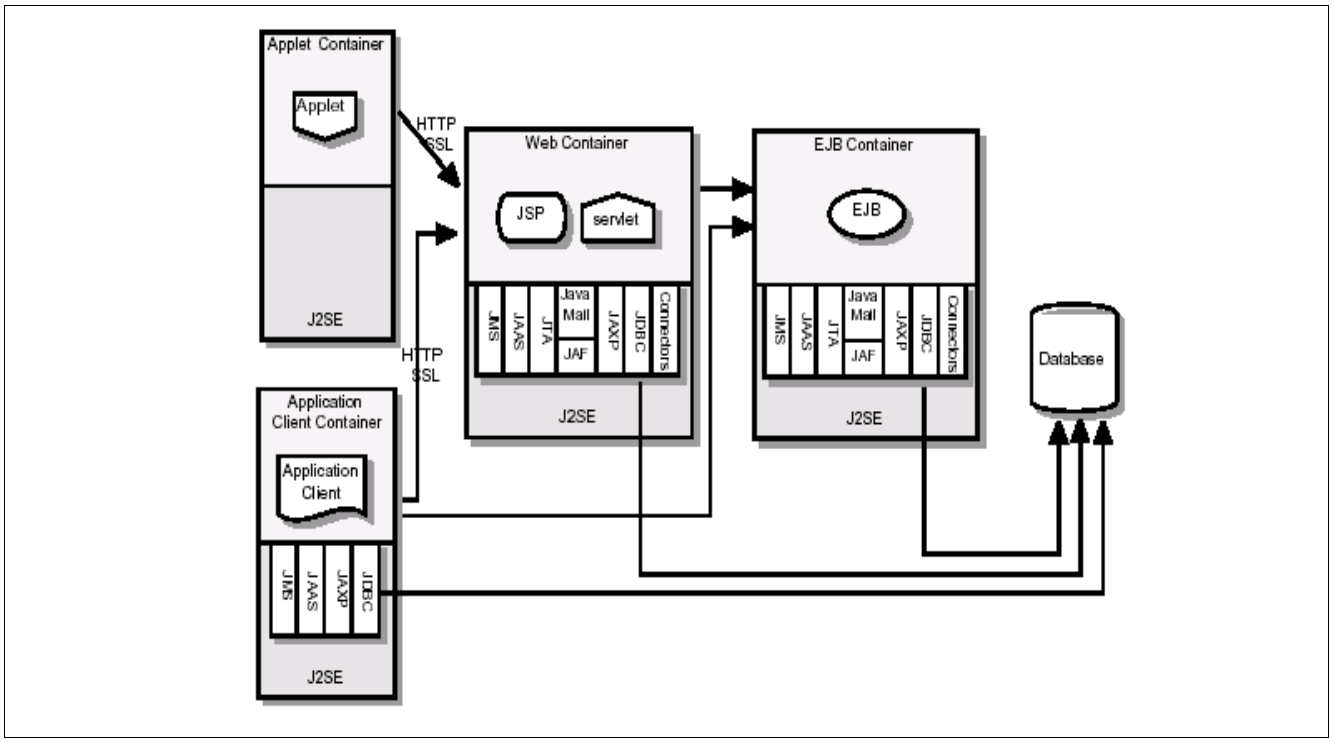


FIGURA I.1: Architettura della J2EE

Questa figura mostra la relazione logica tra gli elementi; questo non significa che corrisponda ad un effettivo partizionamento fisico degli elementi in macchine separate, processi, spazi di indirizzamento o virtual machines.

I contenitori denotati da rettangoli separati sono i runtime enviroment che forniscono i servizi richiesti all'applicazione (componenti rappresentati nella metà superiore dei rettangoli).

Vediamo quali sono le caratteristiche ed i benefici di questa piattaforma innovativa.

Il problema attuale è quello di dover gestire transazioni di molteplici utenti così da poter ridurre i costi e i tempi di risposta, permettendo un accesso semplice ai diversi tipi di funzionalità offerti dall'applicazione. Generalmente le applicazioni che permettono di fornire servizi di questo tipo devono integrare degli EIS (Enterprise Information Systems) già esistenti con nuove applicazioni che permettono di fornire diverse funzionalità che devono poter essere messe a disposizione di un numero elevato di utenti.

Questi servizi messi a disposizione dall' applicazione devono presentare una serie di caratteristiche che vediamo:

- *Versatile:* per poter soddisfare le diverse esigenze degli utenti che fanno uso di questi servizi

- *Sicuro:* per poter proteggere la privacy degli utenti e l'integrità dei dati trasmessi;
- *Riutilizzabile e scalabile:* per poter realizzare applicazioni che possano essere recuperate e riutilizzate da diversi utenti.

Questo modello per la realizzazione di applicazioni fornisce i benefici legati Write Once, Run Anywhere che equivale a dire che le applicazioni multi-livello sono portabili e scalabili.

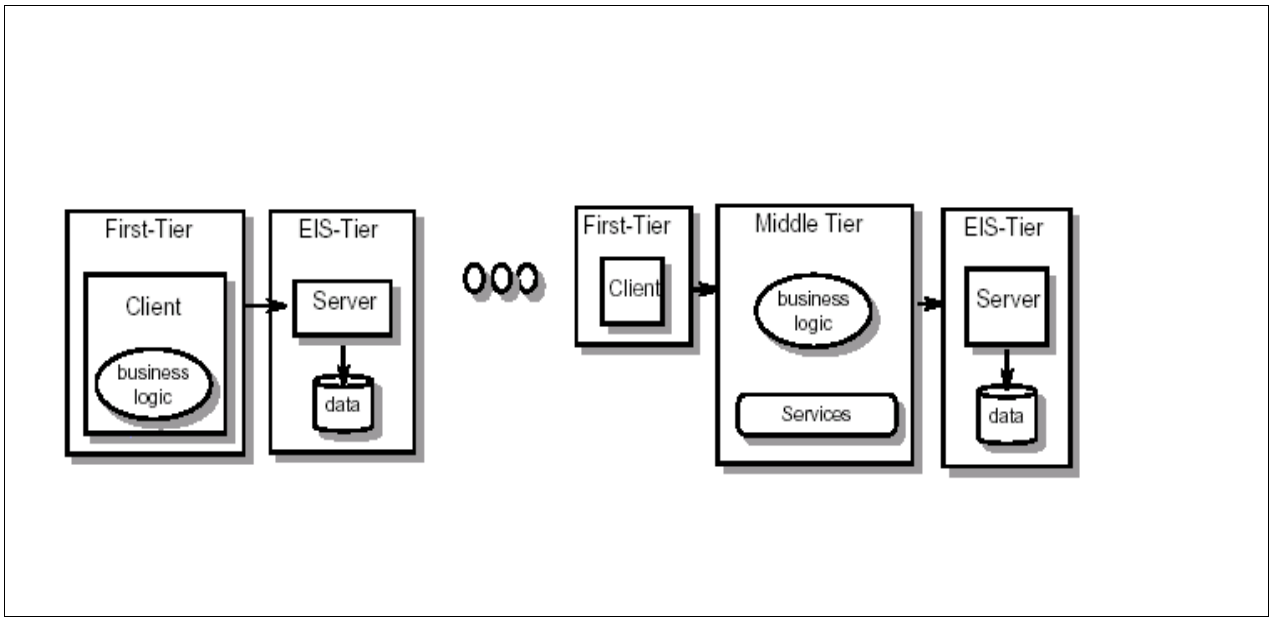


FIGURA I.2: Passaggio da una struttura a due livelli ad una a tre livelli

Questi servizi vengono forniti per mezzo di applicazioni distribuite costituite da diversi livelli caratterizzati all'estremità superiore da un numero elevato e variegato di utenti alla estremità inferiore da diversi tipi di risorse-dati e tutto quello che riguarda il lavoro di sviluppo della applicazione.

Il livello intermedio (definito Middle-Tier) contiene tutti quei servizi che sono in grado di integrare gli EIS preesistenti con nuove applicazioni e risorse-dati.

L'introduzione di questo livello intermedio protegge l'utente dalla complessità connessa con lo sviluppo della applicazione permettendogli un accesso facile e rapido al servizio.

Solitamente il Middle-Tier risiede su un software dedicato ed ha accesso a tutte le risorse messe a disposizione del sistema in modo tale da poter usufruire di tutto quello che è necessario per un corretto funzionamento del programma.

Al contrario tutti i dati critici e le funzioni che li gestiscono risiedono nell'EIS-Tier e rappresentano il nucleo più interno del sistema.

Cerchiamo ora di capire perché si è preferito una piattaforma Multi-Tier ad una Two-Tier.

Inizialmente dovendo gestire delle applicazioni client-server questo tipo di strutturazione permetteva funzionalità e scalabilità precedentemente non disponibili.

Questa struttura però ha diversi limiti che sono rappresentati dal dover inviare a ciascun utente i servizi EIS e quelli di installare e mantenere efficiente, aggiornandola, la parte di applicazione logica sulle diverse macchine utilizzate da diversi utilizzatori.

Per ovviare a questo problema si è pensato di passare ad una struttura a più livelli che oltre ad ovviare agli inconvenienti precedentemente espressi permette di incrementare l'accessibilità che viene richiesta dai vari elementi che costituiscono l'applicazione.

Questa nuova idea ha portato a concentrare l'attenzione sullo sviluppo del software residente nel Middle-Tier.

Questa ristrutturazione ha portato con sé non solo dei vantaggi ma anche delle difficoltà. È stato, infatti, necessario sviluppare separatamente le funzioni di sviluppo per soddisfare i bisogni degli utilizzatori e creare un codice di infrastruttura più complesso in modo da poter accedere ai diversi database e alle diverse risorse di sistema.

Il Modello della J2EE definisce un architettura che permette quindi di fornire servizi che vengono realizzati per mezzo di applicazioni Multi-Tier in grado di sopperire a queste difficoltà e di raggiungere la scalabilitá e la maneggevolezza richiesta.

Il modello implementativo della J2EE richiede di partizionare le applicazioni strutturate su più livelli in due parti:

- ***Business logic***
- ***Presentation logic***

Per business logic si intende quella parte dell'applicazione che riguarda tutte le funzionalità richieste dall'utente mentre la presentation logic rappresenta il modo attraverso il quale l'utente può richiedere i diversi servizi e può osservare i risultati ottenuti.

Nello sviluppare le proprie applicazioni lo sviluppatore può contare sulle funzionalità offerte dalla piattaforma per risolvere i problemi che insorgono nello sviluppare l'applicazione a livello intermedio.

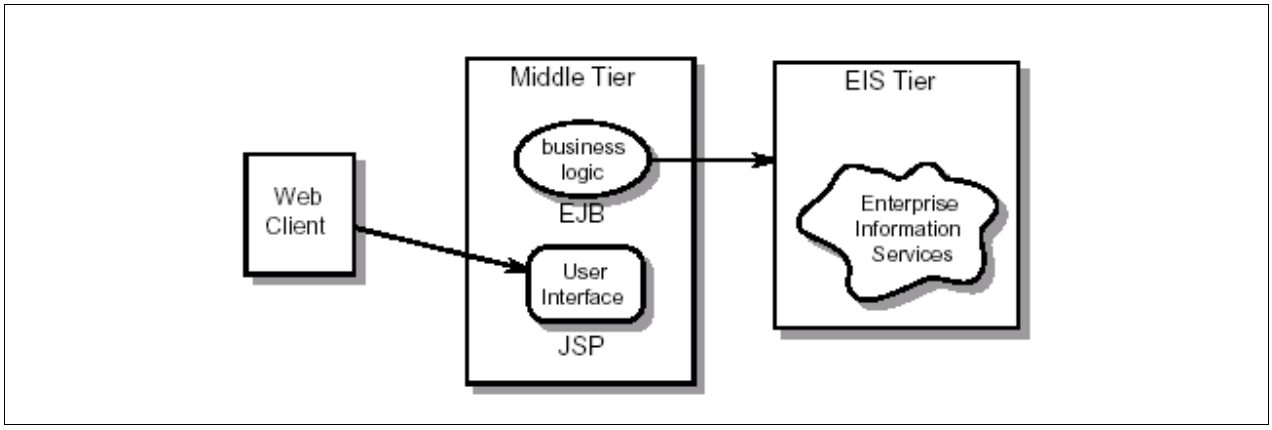


FIGURA I.3: Applicazione a tre livelli

Nella piattaforma J2EE le funzioni di lavoro di questo livello sono sviluppate per mezzo degli EJB (Enterprise Java Beans) come mostrato in figura I.3. Questi EJB permettono agli sviluppatori del servizio di concentrarsi sulla business logic lasciando al EJB server l'onere di occuparsi di tutto ciò che riguarda la gestione del servizio.

Le diverse funzionalità offerte all'utente sono rese disponibili, per mezzo dell'utilizzo della tecnologia che si basa sulle JSP e Servlet, secondo servizi in forma Internet Style facilmente accessibili a qualsiasi utente.

La tecnologia delle JSP (Java Server Pages) permette di rendere molto semplice, per uno sviluppatore di interfacce utente, realizzare pagine dinamicamente generate per qualunque tipo di brower. Le Servlet consentono ai programmatori che, fanno uso di applicazioni basate su linguaggio Java, la possibilità di realizzare presentazioni dinamiche completamente realizzate in Java.

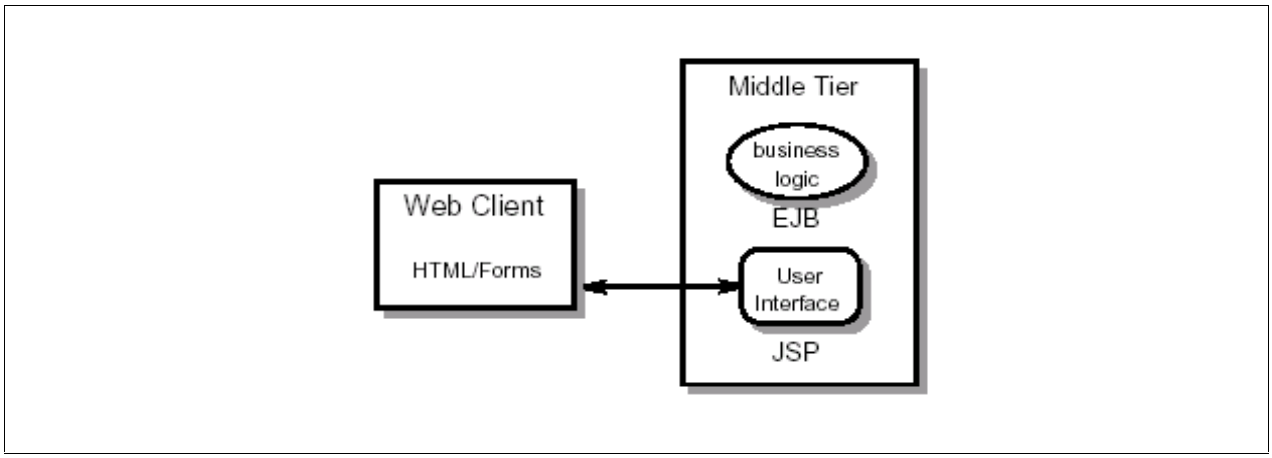


FIGURA I.4: Relazione fra web client e middle-tier

È spesso utile fornire direttamente funzionalità al client che aiuta l'utente ad organizzare e interagire con informazioni del servizio.

In questi casi il servizio scambia il contenuto delle righe con il client invece che con le pagine HTML.

Questo contenuto è spesso nella forma di documenti XML che sono scambiati tra il client ed il sevizio usando il protocollo HTTP. Tipicamente il contenuto di questi documenti XML sono trattati nel First-Tier da componenti Java Beans che sono forniti dal servizio in una applet scaricata automaticamente nel brower dell'utilizzatore come mostrato in figura I.5. In questo caso si parla di HTTP CONTENT BASED CLIENTS.

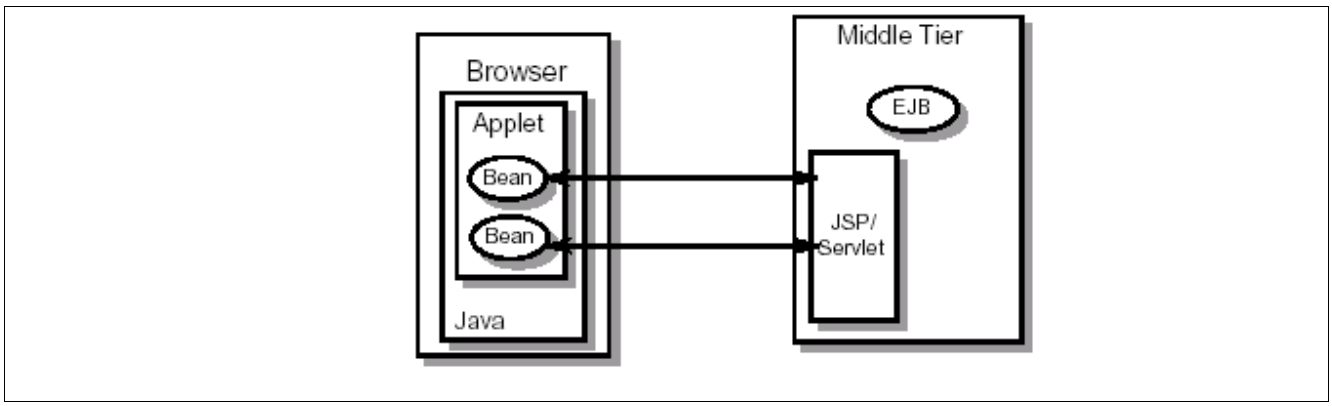


FIGURA I.5: Http content based client

1.3.1.1 Assemblaggio e deployment di un'applicazione J2EE

Un'applicazione J2EE è assemblata in una o più unità standard mentre il deployment può essere realizzato su un qualsiasi sistema che supporta questa piattaforma.

Ogni unità contiene un componente funzionale (EJB, JSP, Servlet, Applet, ...)e un deployment standard che descrive il loro contenuto e contiene le dichiarazioni che sono state specificate dagli sviluppatori dell'applicazione.

Una volta che un'applicazione J2EE è stata prodotta è pronta per subire il deployment su una piattaforma J2EE.

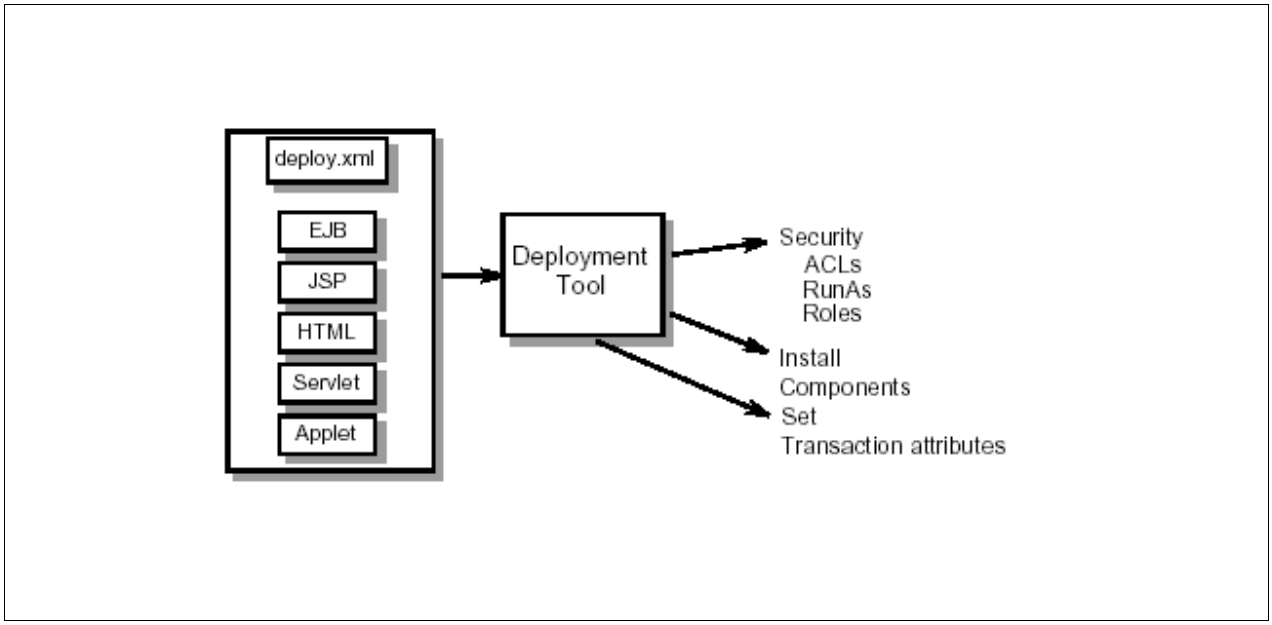


Figura I.6: Deployment di un'applicazione

Questo deployment specifica ad esempio la locazione specifica dell'informazione, la lista degli utilizzatori locali che possono accedere ed il nome del database locale.

Una volta compiuto il deployment dell'applicazione sulla piattaforma locale l'applicazione è pronta per lavorare.

I componenti utilizzati in questa piattaforma sono assemblati in un primo momento separatamente e poi vengono tutti quanti assemblati in una sola applicazione pronta per poter eseguire il deployment. Ogni componente, i suoi file relativi come file gif e html o classi di utilità lato server, ed un deployment descriptor sono assemblati in un modulo e aggiunti ad una applicazione J2EE. Un'applicazione J2EE risulta essere composta da uno o più enterprise beans, componenti Web o moduli

del componente dell'application client. Si deduce da ciò che un'applicazione J2EE può essere costituta da diverse applicazioni J2EE più' semplici tutto dipende da come si decide di organizzare il progetto.

Un deployment descriptor è un documento xml con un estensione .xml che descrive i settaggi del deployment del componente. Il deployment descriptor del modulo di un enterprise bean ad esempio dichiara i suoi attributi e le autorizzazioni per accedere ad esso. Poiché il deployment descriptor è dichiarativo, può essere modificato senza modificare il codice sorgente del bean. Al momento del run-time il server della J2EE legge il deployment descriptor e agisce seguendo le sue direttive. Un applicazione J2EE con tutti i loro moduli viene spedita come un Enterprise Archive (EAR file). Un file EAR è un file di archivio standard di Java con estensione .ear. che contiene al suo interno:

- *Un file Ejb-Jar*
- *Un file Web-War*

Ogni file Ejb-Jar contiene a sua volta un deployment descriptor, i file degl'enterprise beans ed tutti i suoi file relativi. L'archivio dell'applicazioni web contiene i componenti web server-side, classi di utilità (database beans, ecc.), componenti web statici (HTML, immagini, ecc.), classi client-side (applets e loro classi di utilità).

Un file di questo tipo é caratterizzato da una precisa struttura che presenta un direttorio all'interno del quale sono immagazzinate le JSP, le classi client-side ed archivi, e componenti web statici; da un sottodirettorio chiamato WEB_INF che contiene il file web.xml (web application deployment descriptor), i tag library descriptor file; da un'altro sottodirettorio che contiene le classi server-side come servlet, classi di utilità ed i componenti dei java beans ed infine da un altro sottodirettorio chiamato Lib che contiene i file jar delle librerie come tag libraries o ogni altra libreria di utilità che viene richiamata dalle classi server-side.

Con questo modo di procedere si rende possibile il creare un numero differente di applicazioni J2EE riutilizzando alcuni degli stessi componenti. Nessun codice aggiuntivo è richiesto.

Questo formato assicura molteplici benefici:

- *Sicurezza*
- *Minore tempo di download:* se l'applet è assemblata in un file Ejb-Jar, i file delle classi che si riferiscono all'applet e le risorse associate possono essere scaricate da un brower in una singola transazione HTTP senza il bisogno di aprire una nuova connessione per ciascun file.

- *Compressione:* questi file jar permettono di comprimere i diversi file di lavoro per un efficiente immagazzinamento.
- *Portabilita'* : il meccanismo per gestire i file Ejb-Jar è una parte standard della java piattaforma core API.

1.3.1.2 Sicurezza

Contrariamente a quanto richiedono altri modelli di sviluppo in cui è necessaria una piattaforma specifica per gestire la sicurezza in ogni applicazione, i vincoli che riguardano la sicurezza nella J2EE vengono gestiti al momento del deployment time.

Questa piattaforma definisce una serie di regole standard per l'accesso al controllo dell'applicazione in modo tale che il programmatore può definirle anche dopo aver sviluppato il codice sorgente, sarà poi la piattaforma che si preoccuperà di interpretarle al momento del deployment.

Ad esempio la J2EE fornisce una serie di gestori di sistema che permettono di supportare meccanismi di login standard così che le singole applicazioni non devono incorporare nel proprio codice questi meccanismi ma li devono solo richiamare.

Questa soluzione è molto interessante perché permette di utilizzare in diversi momenti vincoli differenti senza dover ogni volta modificare il codice sorgente.

Vediamone un esempio: un sviluppatore specifica diversi livelli di sicurezza (utente generico, super utente, amministratore, ecc.), preoccupandosi solo di scrivere il codice che permette di assegnare un certo utente ad un determinato livello.

Sarà poi il sistema sulla base dei vincoli fissati ad assegnare a ciascun gruppo di appartenenza il livello di competenza permettendo così all'applicazione di verificare i permessi prima che si possa accedere alle operazioni sulle quali sono poste delle restrizioni.

1.3.1.3 Containers

I containers forniscono il supporto run-time per i componenti delle applicazioni della J2EE.

I componenti J2EE dell'applicazione non interagiscono direttamente fra di loro; essi fanno uso dei protocolli e dei metodi del container per interagire l'uno con l'altro e con i servizi della piattaforma.

Il fatto di interporre un container tra i componenti dell'applicazione ed i servizi della J2EE permette al container di far riferimento ai servizi definiti dai deployment descriptors dei componenti.

Un tipico prodotto della J2EE fornirà per ogni tipo di componente dell'applicazione un container: application client container, applet container, web component container, enterprise bean container.

1.3.2 Strumenti utilizzati

Vediamo dapprima schematicamente quali sono stati gli strumenti utilizzati per realizzare quest'applicazione fornendo successivamente qualche nota su ciascuno di essi.

Linguaggio di programmazione:	Java 2
Sistema operativo:	Unix
WEB server:	Tomcat
EJB server:	JBoss
RDBMS:	SQL server

Il linguaggio di programmazione scelto è Java 2 della Sun Microsystem, una piattaforma moderna e perfettamente adatta allo sviluppo di applicazioni web aperte e flessibili e di prestazioni soddisfacenti.

Il sistema operativo sul quale si appoggia l'applicazione è Unix, in quanto fornisce maggiore sicurezza di protezione dei dati e più controllo sugli accessi degli utenti.

Il RDBMS (Relational DataBase Management System) scelto è stato SQL server. Quello che è interessante notare è che il linguaggio di programmazione Java mette a disposizione per la connettività agli RDBMS i driver JDBC che rappresentano uno standard. Questo permette di poter rimanere il più possibile indipendenti dal RDBMS scelto permettendo in un secondo momento, se lo si desidera, di passare ad un altro RDBMS.

Come abbiamo avuto modo di spiegare più diffusamente nei paragrafi precedenti un'applicazione web basata sulla tecnologia J2EE è costituita da una parte che è la business logic ed un'altra che è la presentation logic. Per realizzare la business logic si è scelto JBoss mentre per la presentation logic la scelta è caduta su Tomcat che, in base alle sue caratteristiche, può essere integrato con JBoss in modo tale che possano lavorare assieme.

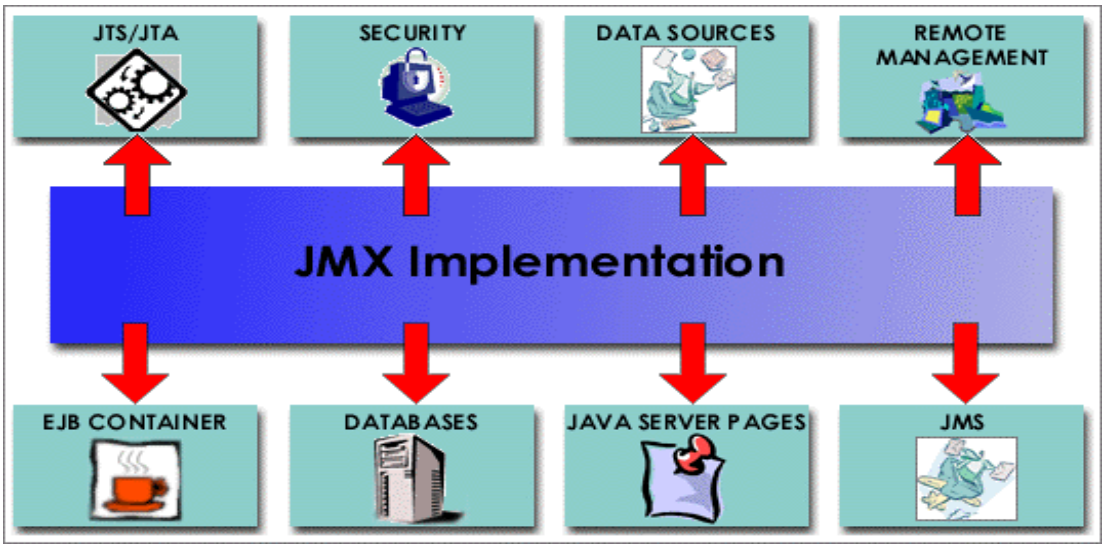
1.3.2.1 JBOSS

JBoss è un EJB server . JBoss è scritto interamente in java e richiede un sistema java compatibile con la jdk 1.3; questo è essenziale non opzionale.

Jboss fornisce jbossserver, l'ejb container di base e l'infrastruttura JMX, jbossMQ for jms massaging, jbossMX per la mail, jbossSX per la sicurezza jass based, jbossCX for la connettivitá jca e jbossCMP per la persistenza CMP.

Jboss permette di miscelare e fondere questi componenti attraverso JMX sostituendo ciascun componente si desideri con un'implementazione JMX conforme alle stesse API. JMX o Java Management Extension è uno strumento ottimo per l'integrazione del software; rappresenta una struttura universale nella quale si possono connettere moduli, containers e plugins.

Figura I.6: Implementazione di JMX



Esiste un database costruito internamente per la gestione di beans persistenti, e questo parte automaticamente. Una delle caratteristiche migliori di jboss è il suo supporto per l'"hot deployment". Questo significa che fare il deploying di un bean equivale a porre il suo file jar nel direttorio di deployment. Se questo viene fatto mentre un bean è caricato , JBoss automaticamente lo interrompe e carica la nuova versione.

Vediamo di seguito come risulta essere strutturato JBoss:

- bin: contiene tutti i file eseguibili (sia scripts che jars) per far partire il server con un Batch (MS-Windows) or Shell (unix) script.

- lib e lib/ext: queste due librerie contengono librerie java in formato jar che JBoss usa. Il direttorio lib contiene jars che devono essere contenuti nel classpath; jars contenuti nella lib/ext sono resi disponibili automaticamente al server.

Se c'è necessità di aggiungere alcune librerie a JBoss, come jdbc driver jars, per esempio, questo può essere fatto inserendo nella direttorio lib/ext così che JBoss li possa utilizzare automaticamente.

- db: direttorio che contiene altri direttori con files relativi al Hypersonic e Instantdb databases (configurazione, indexing, tabelle, etc.).
- deploy: questa è il direttorio del deployment. Basta porre i file jar e ear e di questi verra fatto automaticamente il deployment.
- log: i log file di JBoss sono localizzati in questo direttorio. I file logging sono accesi automaticamente.
- conf: i file di configurazione sono localizzati qui. Di default c'è un solo set di configurazione a disposizione, che è localizzato nel sottodirettorio di default.

Aggiungere più di una configurazione è consentito. Il pacchetto di installazione di jboss con un web container (come tomcat o jetty) crea un ulteriore set di configurazione.

- client: questo è il direttorio dove sono localizzate le librerie per il client.
- docs: contengono la documentazione delle API di JBoss e altra documentazione in formato html.
- external: contiene le librerie jboss.util.jar e metadata.jar.
- src: sotto questa direttorio si trovano l'albero completo dell classi java, in forma sorgente, che fanno funzionare JBoss.
- tmp: questo è un direttorio di lavoro usato dal AutoDeployer per immagazzinare files temporanei.

La configurazione di default di JBoss è localizzata nel direttorio conf/default. Jboss consente all'amministratore di mantenere più di un set di configurazioni. Tutto quello che c'è da fare è di copiare tutti i files della configurazione di default in un nuovo sottodirettorio di conf. Si può procedere poi alla sua modifica in modo da avere diversi tipi di configurazione. L'unica cosa da fare al momento del lancio sarà quello di segnalare a quale configurazione si vuole far riferimento.

Vediamo di seguito quali sono i file di configurazione di JBoss fornendo per ciascuno una qualche nota:

- jboss.properties: è un file nel formato standard dei file java properties che viene caricato nel sistema che gestisce le proprietà al momento della partenza di Jboss.
- jboss.conf: questo file contiene solitamente soltanto quei core Mbeans che sono necessari per il raggiungimento dell'iniziale bootstrap di JBoss come il classpath extension inclusion mechanism, jogging, configuration service, service control.
- jboss.jcml: questo file elenca tutti quei JMX service MBeans che devono essere inclusi nell'instanza funzionante di JBoss.
- jboss-auto.jcml: JBoss ha una caratteristica molto potente che è quella di memorizzare un istantanea runtime detta “snapshot” di tutti gli MBean che stanno girando, inclusi i loro attributi, e di poterla riprodurre dopo in un'altra istanza di JBoss. Quando questa istantanea viene fatta questa viene immagazzinata nel jboss-auto.jcml file.
- mail.properties: JBoss fornisce la possibilità di accesso a risorse del mail provider, come trovare il SMTP e POP servers, come altri settaggi mail-related configuration.
- jnp.properties and jndi.properties: questi due file sono relativi al JNDI. Il primo contiene proprietà di JNP, l'implementazione del provider di JNDI mentre il secondo specifica le proprietà per i client JNDI. Jndi client possono leggere proprietà ascoltate nel jndi.properties file come appare da qualche parte nel suo classpath.
- standardjaws.xml: rappresenta una configurazione di default per il motore CMP di JBoss; contiene il nome jndi del datasource di default, i mappaggi per database jdbc Object-SQL.
- auth.conf: questo file è un jaas login module file, supportato dalla javax.security.auth.login.Configuration.

	L'implementazione contiene solamente le configurazioni server-side di autenticazione che sono applicabili quando si usa JAAS based security.
- server.policy:	è il default java2 security policy per jboss server.
- standardjboss.xml:	questo file fornisce la configurazione di default del container.

1.3.2.2 TOMCAT

Tomcat è un servlet container che viene usato in collaborazione con Jboss per la gestione delle JSP e Java Servlet.

Vediamo come è strutturato Tomcat:

/bin:	contiene i file di startup, di shutdown, file binari eseguibili e scripts.
/conf:	contiene i file di configurazione e le relative DTDs. Il file più' importante per la configurazione del container è il server.xml.
/logs:	contiene i log files che posti qui di default
/webapps:	contiene le applicazioni.
/classes:	contiene le classi globali che vengono utilizzate dalle applicazioni web.
/common:	contiene le classi rese disponibili sia per le classi interne di catalina e per le applicazioni web
/classes:	contiene le classi common.
/lib:	contiene le classi common nel formato dei file jar.
/conf:	contiene i file di configurazione.
/jasper:	contiene i file jar visibili soltanto dal jasper classloader
lib/:	classi globali in formato jar utilizzate dalle applicazioni web.
logs/:	rappresenta il direttorio di destinazione per i file di log.
server/:	contiene le classi interne di catalina e le loro dipendenze.
classes/:	classi interne
lib/:	classi interne in formato jar

webapps/: rappresenta il direttorio base contenente le applicazioni web incluse con Tomcat 4.0

work/: rappresenta un direttorio usato da Tomcat per trattenere file temporanei e direttori

Capitolo 2
Enterprise Java Bean e Java Server Page

Nel primo capitolo abbiamo parlato del fatto che gli strumenti sui quali si basa un'applicazione J2EE sono: EJB (Enterprise Java Beans) e JSP (Java Server Pages).

In questo capitolo tratteremo le caratteristiche di base degli EJB e delle JSP.

Gli EJB rappresentano un nuovo strumento grazie al quale si possono realizzare applicazioni distribuite. È importante notare che già in passato erano disponibili tecnologie che permettevano di raggiungere il medesimo obiettivo anche se risultano essere molto meno efficienti degli EJB. Ad esempio si possono citare RMI (Remote Method Invocation) e CORBA.

Prima di entrare in dettaglio nelle caratteristiche degli EJB è bene soffermarsi su quali sono gli aspetti di base che li caratterizzano:

- ***TRANSAZIONI:*** intendendo con ciò l'insieme di tutte quelle operazioni che devono essere svolte singolarmente ma nel caso si abbiano problemi devono portare al fallimento dell'intero processo;
- ***SICUREZZA:*** avendo a che fare in questo caso con sistemi distribuiti questo è un aspetto di primaria importanza;
- ***SCALABILITÁ:*** questo componente si prefigge l'obiettivo di poter realizzare applicazioni indipendentemente dal numero di utenti che possono accedervi e dalla quantità di dati che ogni volta vengono trasmessi.

Come abbiamo avuto modo di dire in precedenza con l'utilizzo degli EJB ci si propone di realizzare la cosiddetta business logic delle applicazioni strutturate su più livelli in modo tale che i componenti realizzati siano riutilizzabili.

I componenti di base degli EJB sono:

- ***SERVER EJB:*** questo server ha lo scopo di incapsulare tutto quello che sta sotto lo strato EJB (applicazioni legaci, servizi distribuiti) e di fornire ai contenitori gli importanti servizi di base per i vari componenti installati.
- ***CONTENITORE:*** compito fondamentale del contenitore è quello di effettuare controlli di alto livello grazie al fatto che è in grado di intercettare le invocazioni dei client sui metodi dei bean così facendo offre alcune funzionalità legate al life cycle dei vari componenti, alla gestione delle transazioni ed al security management. La corretta comunicazione fra componente e container deve sottostare al fatto che il componente nella sua implementazione deve prevedere alcune interfacce standard.
- ***CLIENT:*** il client rappresenta l'utilizzatore finale del componente. La visione che ha del componente è resa possibile grazie a due interfacce la cui implementazione è effettuata dal container al momento del deploy del bean. La Home Interface fornisce i metodi relativi alla creazione del bean, mentre la Remote Interface offre al client la possibilità di invocare da remoto i vari metodi di business logic. Implementando queste due interfacce, il container è in grado di intercettare le chiamate provenienti dal client e al contempo di fornire ad esso una visione semplificata del componente. Il client non ha la percezione di questa interazione da parte del container. In questo modo si ottiene flessibilità e scalabilitá oltre ad una velocizzazione e semplificazione del lavoro da svolgere per realizzare un componente.

2.1 Classificazione degli Enterprise Java Beans

Gli Enterprise Java Beans (EJB) si possono classificare in 3 diversi tipi:

- ***Entity bean***
- ***Session bean: stateful e stateless***
- ***Message-driven bean***

Lo scopo principale di un entity bean è quello di memorizzare delle informazioni ed offrire al contempo una serie di metodi per la gestione da remoto di tali dati.

Questi sono gli stati che caratterizzano il ciclo di vita di un entity bean:

- ***Stato di pre-caricamento***. Il bean non esiste come entità astratta, ma piuttosto come collezione di file .class e di descriptor files: devono essere fornite al server la primary key, la remote interface e la home interface, oltre a tutti i file generati in modo automatico dal deploy.
- ***Pooled state***. Nel momento in cui il server parte, questo carica in memoria tutti i beans di cui sia stato effettuato correttamente il deploy, posizionandoli nel pool dei bean pronti. In questo momento il componente è disponibile per essere utilizzato quando ci sarà bisogno di servire le richieste del client. Nessun bean in questo caso è assegnato ad un EJBObject, e non contiene informazioni.
- ***Ready state***. Il passaggio a questo stato può avvenire o per esplicita chiamata di un metodo di ricerca da parte del client o per creazione diretta per mezzo dell'invocazione del metodo create.

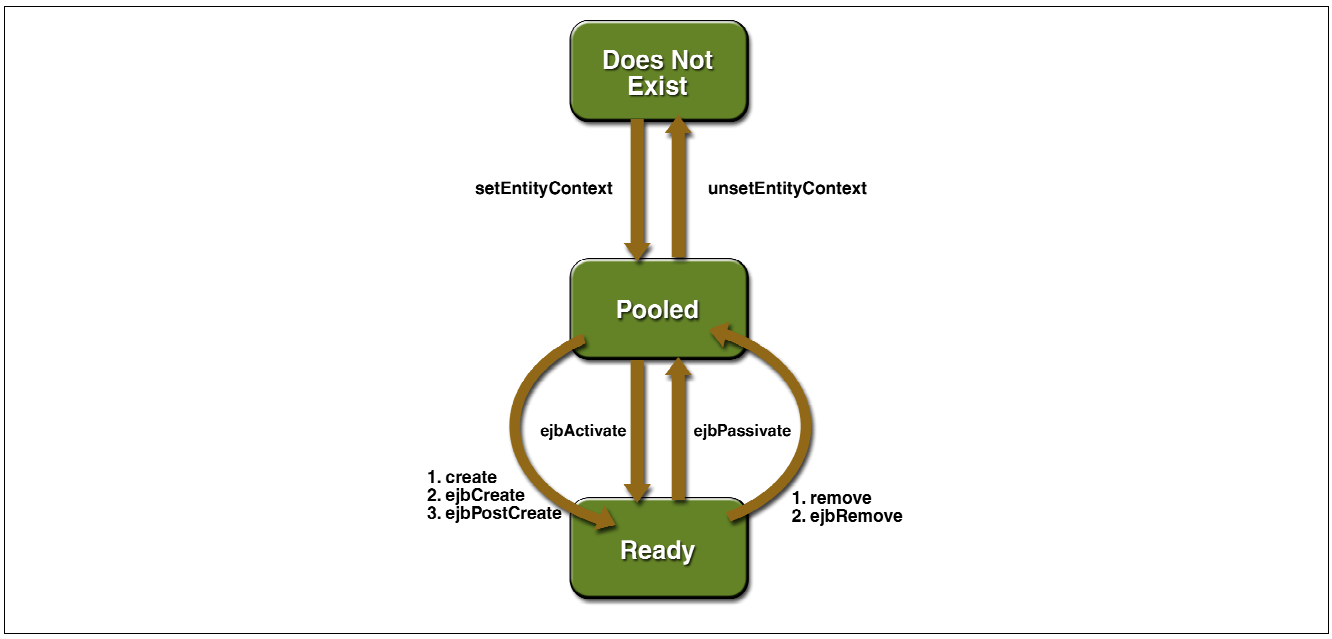


Figura II.1: Ciclo di vita di un entity bean

I session beans sono componenti remoti manipolabili dal client secondo le note home e remote interface che incorporano al loro interno delle funzionalità e servizi. Esistono due tipi di session bean: stateless e stateful. I primi sono componenti che non offrono nessun mantenimento dello stato delle variabili remote fra due invocazioni successive dei metodi da parte del client, i secondi invece si. Tale separazione fra client e bean remoto impone quindi che tutti i parametri necessari al metodo per svolgere il suo compito debbano essere passati dal client stesso al momento dell'invocazione. Uno stateless è spesso visto come un componente che raccoglie alcuni metodi di servizio. Gli stateful sono oggetti lato server funzionanti come contenitori della business logic del client. Questo significa che un determinato bean servirà per tutta la sua vita lo stesso client, anche se il server manterrà sempre attivo un meccanismo di swap fra le varie istanze virtuali dello stesso bean. Essendo dedicato a servire uno e sempre lo stesso client, non insorgono problemi di accesso concorrente. La differenza fra stateless e stateful è che il primo è una raccolta di metodi di servizio, mentre il secondo invece rappresenta l'agente sul server del client.

Gli stati che compongono il ciclo di vita di uno stateless bean sono soltanto due:

- ***Not-exist state***: corrisponde alla non esistenza di un componente all'interno del container.
- ***Ready-pool state***: corrisponde allo stato in cui il bean è pronto per l'invocazione da parte del client. Un bean entra nel ready-pool su richiesta del container al momento del bisogno. Quando il bean si trova nello stato ready-pool è pronto per servire le richieste dei vari client: quando uno di questi invoca un metodo remoto dell'EJBObject relativo, il container associa alla interfaccia remota un bean qualsiasi prelevato dal pool per tutto il periodo necessario al metodo di svolgere il suo compito. Al termine dell'esecuzione il bean viene dissociato dal bean, in netta contrapposizione con quanto avviene con gli entity bean che con gli stateful, dove il bean resta associato allo stesso EJBObject per tutto il tempo in cui il client ha bisogno di interagire con il bean: questo si ripercuote positivamente sulle prestazioni complessive come sulla quantità di memoria occupata dal sistema.

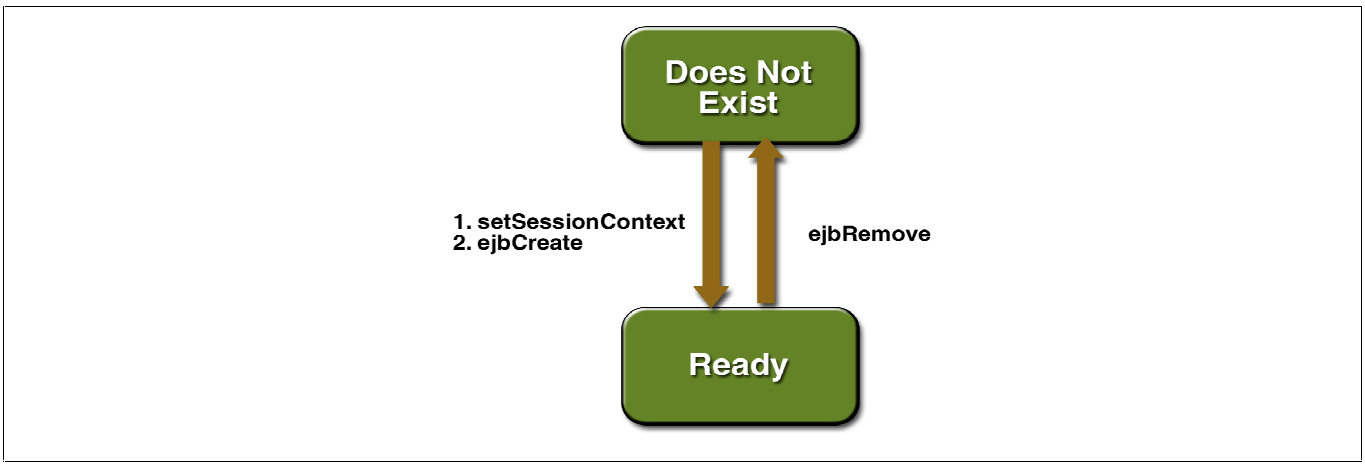


Figura II.2: Ciclo di vita di un session bean stateless

Gli stati che compongono il ciclo di vita di uno stateful bean sono:

- ***Not-exist state*****:** corrisponde allo stato precedente all'istanziazione quando il bean è rappresentato da un insieme di file sul file system.

- ***Ready state*:** alla fase ready si passa per opera del container che invoca il metodo Create(); stato nel quale è libero di rispondere alle invocazioni da parte del client, di accedere ad esempio a risorse memorizzate sul database, o di interagire con altri bean.
- ***Passivated state*:** il passaggio a questo stato può avvenire dopo un periodo più o meno lungo di inutilizzo da parte del client. In questo casi viene rimosso dalla memoria principale (ready state) e reso persistente tramite un qualche meccanismo dipendente dal server. Infine se il componente va in time out durante lo stato di passivated, semplicemente verrà eliminato così come verranno eliminati tutti i riferimenti alla memoria. Il passaggio da passivated a ready avviene quando un client effettua un'invocazione di un metodo remoto di un bean reso persistente.

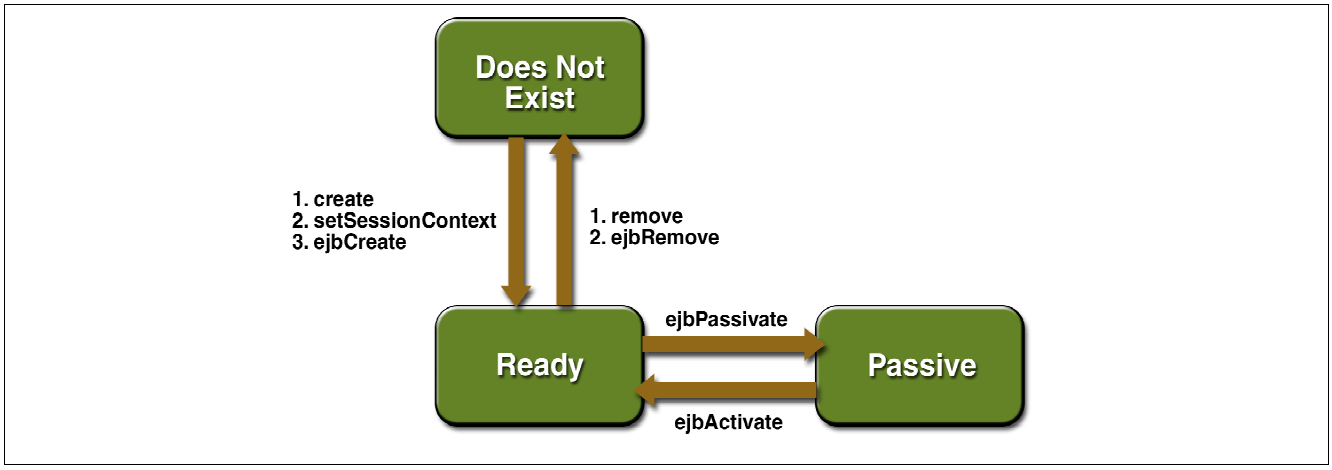


Figura II.3: Ciclo di vita di un session bean stateful

Il message-driven bean agisce come un ascoltatore per i Java Message Service API, processando messaggi in modo asincrono. Questo bean permette alle applicazioni J2EE di processare messaggi in modo asincrono. Egli agisce come un ascoltatore JMS, che è simile ad un ascoltatore di event eccetto che riceve messaggi invece di eventi. I messaggi possono essere spediti da un qualunque componente J2EE (un application client, un altro enterprise bean, o un web component) o da un applicazione JMS o sistema che non usa la tecnologia J2EE. La

differenza più evidente fra i due tipi precedenti e questo è che i client non accedono a questo bean con interfacce. Quest'ultimo ha solo una bean class.

Vediamo le sue caratteristiche:

- Un'istanza di questo bean non trattiene dati o un stato conversational per un cliente specifico.
- Tutte le istanze specifiche sono equivalenti, permettendo all'EJB container di assegnare un messaggio a ciascuna istanza del message bean. Il container può fare il pool di queste istanze per permettere ai messaggi di essere processati in modo concorrenziale.
- Un singolo bean può processare messaggi provenienti da diversi clienti.

Session bean ed entity bean permettono di spedire messaggi JMS e di riceverli in modo sincrono.

Il ciclo di vita di un message-driven bean è caratterizzato dal fatto che l'EJB container solitamente crea un pool di istanze di message-driven bean. Per ciascuna delle istanze, l'EJB container istanzia il bean, chiama il metodo detto setmessageDrivenContext per passare il context object all'istanza, chiama l'istanza del metodo ejbCreate ed infine al termine del ciclo di vita del bean il container chiama il metodo ejbRemove.

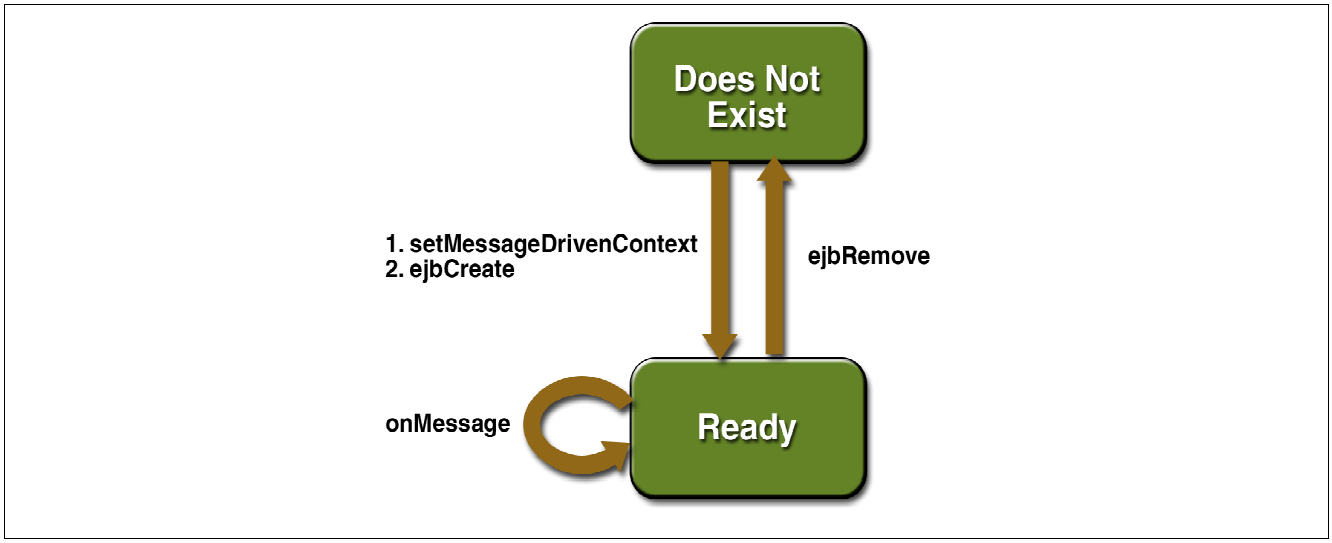


Figura II.4: Ciclo di vita di un message-driven bean

2.2 Struttura di un Enterprise Java Bean

Vediamo ora come viene ad essere strutturato un bean in generale per poi addentrarci nei dettagli di ciascuno di essi.

La struttura base degli EJB è caratterizzata da 3 diverse parti che pur essendo sviluppate separatamente presentano dei legami che se non rispettati portano al non deployment da parte del sistema. La prima parte è rappresentata dalla cosiddetta remote interface che contiene al suo interno tutti quei metodi che potranno essere richiamati dal client. L'oggetto che implementa tale interfaccia e che viene creato al momento del deploy, viene detto in genere EJBObject. La seconda parte è costituita dalla home interface che contiene tutti quei metodi che vengono invocati dal container e che sono necessari per la gestione del ciclo di vita del bean (es. fasi di creazione ed installazione del container, rimozione, operazioni di ricerca per mezzo di chiavi univoche). Questo tipo di interfaccia assume il nome comune di EJBHome. Infine si ha la bean class che contiene la business logic del bean. Questa classe appena definita non dovrà contenere al suo interno dei riferimenti alle altre due interfacce però, naturalmente, ci dovranno essere delle coincidenze fra quello definito nelle due interfacce e quello definito in questa classe. Mi spiego meglio. I metodi definiti nella remote interface dovranno avere nomi e contenere attributi coincidenti con quelli definiti in questa classe. Questo tipo di sviluppo può sembrare complesso e non immediato in quanto richiede di implementare 3 diverse parti caratterizzate da specifiche regole di realizzazione e quindi sembra irragionevole lasciare la strada percorsa prima dell'avvento degli EJB. In realtà da un osservazione più attenta ci si rende conto che questo modo di procedere rende il lavoro molto più semplice per lo sviluppatore. Egli, infatti, una volta create queste 3 parti, con le dovute accortezze, delega tutto il resto al container il quale oltre a consentire l'interazione fra il client e l'oggetto remoto in modo indiretto, si preoccupa di creare nuove istanze di oggetti remoti e di verificarne la corretta installazione ed il giusto funzionamento.

Nella prosecuzione del capitolo si prenderanno principalmente in considerazione i session bean proprio perché essi sono gli unici bean utilizzati all'interno dell'applicazione che è stata realizzata. Verranno presi in considerazione di riflesso anche gli entity perché comunque risultano essere simili ai session bean.

La scelta di utilizzare solo dei session bean è stata dettata dal fatto che l'applicazione per funzionare richiede soltanto dei componenti che assolvano al proprio compito mantenendo alcun

riferimento in memoria. Al loro interno si possono inserire tutte le operazioni che si desidera siano effettuate durante il ciclo di vita del bean, senza che ci si debba preoccupare di quando e come essi verranno invocati, ovvero tralasciando i problemi legati a come e quando verranno effettuati i passaggi di stato sul componente. Così facendo si ha la possibilità di ottenere la netta separazione fra colui che sviluppa il bean e chi sviluppa il server.

All'interno del bean saranno presenti dei metodi sviluppati dallo stesso sviluppatore ma anche dei metodi che sono necessari per la realizzazione stessa dell'EJB.

Vediamo quali sono questi metodi:

- ***Ejbcreate() ed ejbPostcreate():*** sono invocati dal container sul componente rispettivamente subito prima e subito dopo la creazione del componente;

- ***ejbRemove():*** comunica al componente che sta per essere rimosso dal server, e che i dati relativi, memorizzati nel db, stanno per essere cancellati;

- ***ejbLoad() e ejbStore():*** notificano che lo stato del componente sta per essere sincronizzato con il database. In particolare il metodo ejbStore() viene chiamato dal container subito prima che il componente venga memorizzato nel database, e tra le altre cose permette al programmatore di effettuare tutte le operazioni di sincronizzazione e messa a punto prima che il componente stesso sia reso persistente. Analogamente l'invocazione di ejbLoad() avviene immediatamente dopo la fase di caricamento del componente dal db. I metodi

- ***ejbActivate() e ejbPassivate():*** notificano al componente che sta per essere attivato o disattivato.

2.3 Pooling delle risorse

In diverse occasioni si è sottolineato il fatto che si vuole realizzare un'applicazione che possa essere fruibile da un numero anche elevato di utenti. Nel momento in cui si concede la possibilità a diversi individui di poter accedere alla medesima risorsa nasce di conseguenza il problema di come questo accesso debba essere regolamentato. Il meccanismo utilizzato in questo caso, preso in prestito dai sistemi di gestione di basi di dati, è quello del pooling delle risorse. Alla base di questa architettura vi è la considerazione secondo la quale molto raramente si verifica l'ipotesi per cui tutti i client

dovranno accedere nello stesso istante ai vari bean installati sul server: di rado si renderà quindi necessario istanziare e referenziare contemporaneamente tutti gli oggetti remoti che corrispondono ai molti client in funzione. Oltre ad una drastica riduzione del numero di connessioni aperte e di oggetti attivi, il meccanismo del pool di oggetti remoti risulta anche essere più efficiente, infatti, il meccanismo di condivisione risulta essere più efficiente rispetto alla creazione/distruzione sia di connessioni che di bean. In definitiva solo pochi oggetti remoti verranno istanziati ed utilizzati, dando vita ad una politica di condivisione dei bean tramite le molte interfacce istanziate. Questa separazione fra ciò che è in funzione sul lato server e quello che invece il client gestisce è reso possibile grazie al fatto che il client non accede mai direttamente agli oggetti remoti, ma sempre tramite interfacce remote, secondo lo standard di EJB.

Per comprendere come sia messo in atto il sistema di pooling, si prenda prima in esame il caso degli entity bean. Sulla base del ciclo di vita di un entity bean risulta intuitivo capire come viene ad essere gestito il pooling: il server, infatti, istanzia una serie di component remoti, ponendoli nel pool, ed associando un componente remoto ad un determinato EJBObject solo al momento dell'effettivo bisogno. Tutti i componenti remoti sono quindi equivalenti fra loro mentre sono nello stato pooled, ed acquistano una contestualizzazione solo al momento della invocazione da parte del client di uno dei business methods del bean: il client effettua le invocazioni ai metodi dell'EJBObject e non direttamente dell'oggetto remoto. Nella fase di ready il componente riceve le invocazioni in callback dal server e non dal client. Appena il componente viene istanziato e posizionato nel pool, riceve un nuovo context (istanza di un javax.ejb.EJBContext) che offre un'interfaccia al bean per comunicare con l'EJB enviroment. L'EJBContext è in grado sia di reperire informazioni sul client che ha effettuato le invocazioni, sia di fornire un riferimento alle interfacce EJBHome ed EJBObject in modo da permettere l'invocazione da parte del bean stesso dei metodi di business di altri componenti. Quindi il bean remoto viene invalidato, nel caso in cui il client invochi uno dei metodi di rimozione, oppure perchè il componente è uscito dallo scope di esecuzione, il bean viene separato dall'EJBObject e successivamente riposizionato nel pool. Può accadere che un bean sia riammesso nel pool dopo un periodo di inutilizzo prolungato da parte del client: in tal caso, nel momento in cui il client dovesse richiedere nuovamente l'utilizzo del bean, allora verrà effettuata nuovamente un'operazione di assegnazione prelevando dal pool il primo componente disponibile. Questa operazione detta instance swapping è particolarmente importante dato che permette di servire un alto numero di richieste utilizzando poche risorse attive: il tempo in cui il bean viene utilizzato dal client staticamente è

mediamente minore del tempo passato in attesa nel pool. Essendo l'instance swapping una operazione meno costosa della creazione di un bean, giustifica quindi l'utilizzo di un pool di bean. Trattiamo ora il pooling di session bean. Nel caso degli stateless il meccanismo di pool è semplificato dalla natura stessa del componente: in questo caso, infatti, non viene mantenuta alcuna informazione fra due invocazioni dei metodi da parte del client, e quindi non è necessario memorizzare lo stato. Ogni metodo esegue le istruzioni senza che si debba accedere in qualche modo ad informazioni memorizzate da qualche parte.

Questo permette al container di effettuare il processo di swapping senza tener conto di quale particolare istanza venga utilizzata. Nel caso degli stateful session bean invece le cose sono leggermente diverse: l'integrità delle informazioni deve essere mantenuta in modo da ricostruire in ogni momento la storia delle varie invocazioni (che si definisce conversational state , ovvero stato della conversazione fra client e bean); per gli stateful beans quindi non viene utilizzato il meccanismo di swapping di contesto. In questo caso tutte le volte in cui sia necessario utilizzare un componente, è sufficiente prelevarne uno vuoto dalla memoria (quindi in modo simile al sistema di pool), ripristinandone lo stato prelevando le informazioni direttamente da un sistema di memorizzazione secondaria (solitamente file serializzati su file system). Questa operazione viene detta attivazione ed è simmetrica a quella della scrittura su disco che prende il nome di passivazione. I metodi di callback utilizzati dal server per informare degli eventi di attivazione e passivazione sono i due ejbActivate() ed ejbPassivate(): il primo viene invocato immediatamente dopo l'attivazione, ed il secondo immediatamente prima la passivazione del componente.

2.4 La gestione della concorrenza

Veniamo ora ad affrontare il problema della concorrenza. Questo tipo di problema lo si deve però soltanto circoscrivere agli entity bean. Infatti, per la propria natura i session bean non presentano problemi di questo tipo. Mi spiego meglio: ogni componente stateful è associato ad uno ed un solo client quindi il soggetto invocante è sempre lo stesso: nessuna concorrenza; nel caso degli stateless, non memorizzando nessun tipo di stato, due bean non potranno mai interferire fra loro. Veniamo ora a vedere la soluzione adottata per gestire questo problema: l'accesso contemporaneo è limitato non solo a livello di risorsa specifica ma più precisamente di thread di esecuzione. Ogni invocazione di un metodo di un bean avviene sempre e comunque secondo lo schema classico dell'invocazione remota basata su EJBObject. Anche se questa organizzazione può apparire eccessivamente complessa, presenta due

importanti vantaggi: da un lato permette di mantenere un elevato livello di standardizzazione, dato che ogni componente deve essere progettato ed implementato seguendo una rigida specifica standard. Dall'altro, visto che A invoca i metodi di B tramite un interfaccia remota, è possibile effettuare sostituzioni, e migrazioni del bean da un server ad un altro, senza che il client ne abbia percezione. Questo permette massima flessibilità, semplicità di gestione e con poco lavoro consente l'implementazione di tecniche di load balancing fra server differenti. Questo tipo si soluzione crea però anche dei problemi, infatti, ogni invocazione è infatti vista come una invocazione remota alla RMI anche se l'oggetto è in esecuzione localmente.

2.5 La gestione della persistenza

Questo argomento riguarda solo gli entity bean, infatti, essi rappresentano dati o collezione di dati che possono essere memorizzati tramite l'utilizzo di un database di vario tipo. La gestione della persistenza può essere fatto in due modi: container managed o bean managed. Nel primo caso è il container che si preoccupa di fare il mapping e la sincronizzazione fra i dati contenuti nel componente, e quelli memorizzati nel database. Nel secondo caso invece, sempre appoggiandosi ad un archivio esterno, tutta la logica di sincronizzazione dei dati deve essere implementata a mano. Come apparentemente appare ovvio la prima è la soluzione più comoda e sicura, mentre la bean managed viene utilizzata ad esempio quando sia necessaria una personalizzazione del processo di salvataggio dei dati. Il legame fra bean e dati memorizzati nel database è talmente stretto che una modifica nel primo si riflette in un aggiornamento dei dati. Il database da utilizzare per la memorizzazione di un entity può essere di tipo relazionale o ad oggetti. La scelta solitamente cade su quello relazionale grazie alla sua diffusione e affidabilità. Si presenta un problema che essendo esso estraneo alla logica ad oggetti utilizzata negli EJB si rende necessario uno strato software con cui effettuare il mapping relazionale object-oriented, in modo da trasformare i campi di un bean in valori delle colonne di un database. I database ad oggetti non richiedono nessuna operazione di conversione, e quindi permettono ad un componente di essere salvato in modo diretto nel database stesso, consentendo di limitare la complessità della struttura.

2.6 Servizio di Naming

Qualsiasi sistema di programmazione distribuita ha alla base un qualche meccanismo che permetta di rintracciare oggetti remoti in esecuzione in spazi di indirizzamento eterogenei. Un sistema di questo tipo si basa sui cosiddetti naming services che sono essenzialmente di due tipi: servizio di bind (ovvero registrazione di un oggetto remoto in un qualche registry tramite nome logico) e servizio di lookup (che corrisponde all'operazione inversa, ovvero ricercare la particolare istanza di un oggetto remoto, partendo da un nome logico). Nel caso degli EJB il meccanismo utilizzato è quello offerto dalla Java Naming and Directory Interface (JNDI API) questo strumento implementa un'astrazione gerarchica ad alto livello di un'ipotetica collezione di risorse (file, oggetti, devices vari)

2.7 Sicurezza

La sicurezza si può intendere a 3 livelli: autenticazione, access control e sicurezza nella comunicazione. Nel primo caso grazie ad un qualche sistema di riconoscimento si deve consentire a un utente (persona fisica) o entità (un oggetto) l'accesso al sistema e di poter usufruire dei servizi messi a disposizione. Il riconoscimento può avvenire tramite password, certificato digitale o tramite meccanismi particolari come smart card o devices elettromagnetici. Nel caso in cui si disponga di un sistema di access control è possibile definire una policy di permessi in modo da autorizzare un utente ad effettuare determinate operazioni. In questo casi ad ogni client viene associato un elemento di istanza java.security.Identity che rappresenta l'identità con cui il client potrà effettuare le operazioni con gli oggetti remoti. La identity associata ad un client viene utilizzata in modo del tutto trasparente: ad esempio quando il client richiede l'utilizzo di un particolare metodo, all'invocazione dello stesso viene passata anche l'istanza dell'identità associata, al fine di effettuare il controllo sui diritti. La definizione delle policy si effettua al momento del deploy grazie al deployment descriptor il quale contiene varie istanze di ControlDescriptor ed AccessControlEntry: questi ultimi permettono di definire la lista degli utenti ammessi ad effettuare determinate azioni, mentre i ControlDescriptor consentono fra l'altro di specificare il "runAs" di ogni Identity, ovvero con quale identità ogni metodo può essere eseguito.

Lo scopo principale di un entity bean è quello di memorizzare dell informazioni ed offrire al contempo una serie di metodi per la gestione da remoto di tali dati. In questa ottica è possibile suddividere le funzionalità (metodi) di un bean in due categorie: da una parte si trova tutto ciò che è relativo alla gestione da remoto tramite il client, ovvero la business logic, mentre dall'altro si trova ciò

che è relativo alla gestione del ciclo di vita del componente, del mantenimento dello stato e della gestione della sicurezza. Vediamo come gli entity bean implementano la persistenza dei dati all'interno dei database: da un lato il processo di sincronizzazione viene gestito dal container in modo trasparente, mentre nell'altro caso tutte le informazioni sono salvate su disco direttamente tramite operazioni effettuate da bean. Nel primo caso si parla CMP (Container Managed Persistence), mentre nell'altro caso si parla di BMP (Bean Managed Persistence).

2.8 La gestione delle transazioni

L'accesso concorrente da parte di più client sullo stesso set di bean, e quindi ai dati memorizzati nel database, rende necessario un qualche sistema di controllo dei dati in modo tale da impedire configurazioni incoerenti sui dati stessi o sui risultati di tali operazioni. Il modello a cui si fa riferimento è il modello ACID, ovvero una transazione che deve essere atomica (Atomic), consistente (Consistent), isolata (Isolated) e duratura (Durable). Atomic indica che tutte le operazioni che costituiscono la transazione devono essere eseguite senza interruzioni; nel caso dei database se avvengono transazioni che subiscono delle interruzioni dovranno essere riassegnati i valori iniziali precedenti all'inizio della transazione, se tutto va bene le modifiche sui dati del database potranno essere effettuate realmente. La consistenza dei dati si ottiene grazie al lavoro congiunto del sistema transazionale e dello sviluppatore. L'isolamento garantisce che la transazione verrà eseguita dall'inizio alla fine senza l'interferenza di elementi esterni o altri soggetti. Il fatto che sia duraturo infine deve garantire che le modifiche temporanee ai dati debbano essere effettuate in modo persistente in modo da evitare che un eventuale crash del sistema possa portare alla perdita di tutte le operazioni intermedie. Nel caso in cui si voglia utilizzare il motore transazionale del container EJB, si può definire il comportamento del bean tramite gli attributi transazionali. Prima di procedere all'elencazione di questi attributi transazionali diamo la definizione di scope transazionale. Esso rappresenta un insieme di bean che prendono parte ad una determinata transazione. Il termine scope viene utilizzato proprio per dar risalto al concetto di spazio di esecuzione: infatti, ogni volta che un bean facente parte di un determinato scope invoca i metodi di un altro o ne ricava un qualche riferimento, causa l'inclusione di quest'ultimo nello stesso scope a cui appartiene lui stesso:di conseguenza quando il primo bean transazionale prende vita, lo scope verrà propagato a tutti i bean interessati.

Gli attributi transazionali sono:

- ***Not supported:*** invocando all'interno di una transazione di un bean settato con questo valore, si otterrà una interruzione della transazione; lo scope della transazione non verrà propagato al bean o ad altri da lui invocati. Appena il metodo termina, la transazione riprenderà la sua esecuzione.
- ***Supports:*** l'invocazione da parte di un client incluso già in uno scope, provocherà la propagazione di tale scope al metodo. Ovviamente non è necessario che il metodo sia necessariamente invocato all'interno di uno scope, per cui potranno invocarlo anche client non facenti parte di nessuna transazione.
- ***Required:*** si ha la necessità della presenza di uno scope per l'invocazione del metodo. Se il client verrà propagato, altrimenti uno nuovo verrà creato appositamente per il metodo del bean(scope che verrà terminato al termine del metodo).
- ***Requires New:*** il bean invocato entra in una nuova transazione , sia che il client faccia parte o meno di una transazione. Se il client è coinvolto in una transazione, quest'ultima verrà interrotta fino al completamento della transazione del bean invocato. Il nuovo scope creato per bean verrà propagato esclusivamente a tutti i bean invocati dal bean di partenza. Quando il bean invocato terminerà la sua esecuzione, il controllo ritornerà al cient che riprenderà la sua transazione.
- ***Mandatory:*** il bean deve sempre essere parte di una transazione; nel casi in cui il client invocante non appartenga a nessun scope transazionale, il metodo del bean genererà un'eccezione TransactionRequiredException.
- ***Never:*** il client invocante non può appartenere a nessun scope transazionale, altrimenti il bean invocato genererà RemoteException.

Il livello di isolamento di una transazione in genere è valutabile in funzione del modo in cui riesce a risolvere i seguenti problemi:

- ***Dirty reads:*** si immagini il caso in cui due transazioni, una di lettura ed una di scrittura, debbano accedere ai medesimi dati. In questo caso si possono avere incoerenze dei dati, nel caso in cui la transazione in lettura accedesse ai dati appena modificati da quelli in scrittura, quando quest'ultima dovesse per un motivo qualsiasi effettuare un rollback riponendo i dati nella configurazione originaria.

- ***Repeteable reads:*** questa condizione garantisce l'immutabilità dei dati al succedersi di differenti letture all'interno della stessa transazione (ovvero elimina la possibilità che un soggetto possa modificare i dati oggetto della transazione mentre questa è in atto). Una lettura non ripetibile si ha quando una transazione dopo una prima lettura, vedrà alla seguente le modifiche effettuate dalle altre transazioni. In genere questa è garantita o tramite un lock sui dati, oppure tramite l'utilizzo di copie di dati in memoria su cui effettuare la modifiche. La prima soluzione è probabilmente più sicura, anche se impatta pesantemente sulle prestazioni. La seconda invece può complicare molto la situazione, a causa delle difficoltà derivanti dalla necessità di sincronizzare i dati copia con quelli originali.
- ***Phantom reads:*** letture di questo tipo possono verificarsi quando nuovi dati aggiunti al database sono visibili anche all'interno di transazioni iniziate prima dell'aggiunta dei dati al database, transazioni che quindi si ritrovano nuovi dati la cui presenza è apparentemente immotivata.

Ecco alcune soluzioni per ovviare a questi problemi:

- ***Read locks:*** questo blocco impedisce ad altre transazioni di modificare i dati quando una determinata transazione ne effettua una lettura. Questo previene l'insorgere di letture non ripetibili, dato che le tre transazioni possono leggere i dati ma non modificarli o aggiungerne di nuovi. Se il lock viene effettuato su un record, su tutta la tabella oppure su tutto database dipende dalla implementazioni particolare del database.
- ***Write locks:*** in questo caso, che si presenta tipicamente in operazioni di aggiornamento, alle altre transazioni è impedito di effettuare modifiche di dati; rappresenta un livello di sicurezza aggiuntivo rispetto al caso precedente, ma non impedisce l'insorgere di letture sporche dei dati da parte di altre transazioni ed anche di quella in corso.
- ***Exclusive locks:*** questo è il blocco più restrittivo ed impedisce alle tre transazioni di effettuare letture o scritture su dati bloccati: le dirty e phantom reads quindi non possono verificarsi.

- ***Snapshot:*** alcuni sistemi offrono un meccanismo alternativo ai lock, detto comunemente di snapshot (istantanea) dei dati: in questo caso sono create al momento dell'inizio della transazione delle vere e proprie copie dei dati, tali da permettere di lavorare in lettura e scrittura su copia dei dati. Se questo elimina del tutto il problema dell'accesso concorrente, introduce problemi non banali relativamente alla sincronizzazione dei dati reali con le varie snapshot.

Vediamo quali sono i livelli di isolamento delle transazioni:

- ***Read uncommitted:*** una transazione può leggere tutti i dati uncommitted (ovvero quelli ancora non resi persistenti) di altre transazioni in atto. Corrisponde al livello di garanzia più basso dato che può dar vita a dirty e phantom reads, così come possono verificarsi letture non ripetibili.
- ***Read committed:*** una transazione non può leggere i dati temporanei(not committed) di altre transazioni in atto. Sono impedite le dirty reads, ma possono verificarsi le letture fantasma e le non ripetibili. I metodi di un bean con questo livello di isolamento non possono leggere dati affetti da una transazione.
- ***Repetable reads:*** una transazione non può modificare i dati letti da un'altra transazione. Sono impedite le dirty reads e le letture fantasma ma possono verificarsi le letture non ripetibili.
- ***Serializable:*** corrisponde al livello massimo di sicurezza, dato che una determinata transazione ha l'esclusivo diritto di accesso in lettura e scrittura sui dati. Si ha la garanzia contro le dirty reads, le letture fantasma e le non ripetibili.

Sebbene la gestione delle transazioni tramite il motore transazionale del server EJB sia la soluzione di gran lunga più semplice e preferibile, non l'unica possibile. Si possono, infatti, implemetaare tecniche manuali per ottenere risultati analoghi ma con un maggior controllo sulle singole operazioni effettuate. La gestione manuale, detta anche esplicita, delle transazioni si basa in genere su qualche motore sottostante basato sul modello Object Transaction Model(OMT).

2.9 Generazione dinamica di pagine web

Veniamo ora a parlare delle Java Server Page che sono componenti che ci permettono di realizzare la presentation logic. Le JSP (Java Server Pages) rappresentano una tecnologia server side. Esse sono pagine HTML che contengono al loro interno del codice Java. Questo tipo di pagine non esistono come risorse statiche ma vengono generate dinamicamente come risultato di una determinata richiesta da parte del client. Da un punto di vista funzionale una pagina JSP si può considerare come un file di testo scritto secondo le regole di un markup language in base al quale il contenuto del file viene elaborato da un JSP container per restituire il risultato di una trasformazione del testo originale, secondo le istruzioni inserite nel testo. Un JSP container si comporta più come un template processor integrato in un application server che come un semplice markup processor. L' intero processo di generazione del codice, compilazione ed esecuzione è gestito automaticamente in modo trasparente.

Un pagina JSP e' un file in formato testo che comprende essenzialmente due tipi di testo:

- ***Template text:*** "testo letterale" destinato a rimanere tale e quale dopo l'elaborazione della pagina;
- ***JSP text:*** porzioni di testo che vengono interpretate ed elaborate dal JSP container.

Quindi nelle JSP solo alcune parti del testo vengono interpretate ed elaborate, mentre non ci sono restrizioni sul contenuto del template text. In pratica, nella maggior parte dei casi, il template text e' in formato HTML (o a volte XML), dato che le JSP sono pagine web. Dal punto di vista del programmatore java, invece, una java server page può essere vista come un modo particolare per interfacciarsi ad oggetti java, in particolare a servlet e bean. Infatti, il JSP container converte la pagina JSP in una servlet, generando prima il codice sorgente, poi compilandolo. Il servlet così generato potrà

interagire con componenti JavaBeans, secondo le istruzioni espressamente inserite nella pagina JSP. E' necessario compilare il codice per ottenere il byte code eseguibile
dalla virtual machine. Il processo di compilazione sarà lanciato una sola volta, alla prima richiesta, dopodiché sarà utilizzata la classe java già compilata tutte le altre volte. In sostanza si può pensare alle JSP come a un modo per generare automaticamente un servlet ad hoc per la gestione di un pagina web a contenuto dinamico, ossia come a una forma di servlet authoring.

L'esecuzione di una pagina JSP prevede due fasi distinte:

- Fase di traduzione-compilazione, durante la quale viene costruito un oggetto eseguibile dalla Virtual Machine (in genere un servlet) in grado di elaborare una risposta alle richieste implicite o esplicite contenute nella pagina.
- Fase di processing, in cui viene mandato in esecuzione il codice generato e viene effettivamente elaborata e restituita la risposta.

2.10 Elementi che compongono una Java Server Page

Riportiamo di seguito quelli che risultano essere gli elementi principali costituenti una Java Server Page:

- ***Template text:*** tutte le parti di testo che sono definite come elementi JSP, vengono copiate tali e quali nella pagina di risposta.
- ***Comment:*** con sintassi <%-- comment--%>; sono commenti che riguardano la JSP in quanto tale, e pertanto vengono eliminati dal JSP container nella fase di traduzione-compilazione; da non confondere con i commenti HTML e XML, che vengono inclusi nella risposta come normale template text.
- ***Directive:*** con sintassi <%@ directive...%>; sono direttive di carattere generale, indipendenti dal contenuto specifico della pagina, relative alla fase di traduzione-compilazione.

- ***Action:*** seguono la sintassi dei tag XML ossia <tag attributes> body </tag> oppure <tag attributes/> dove attributes sono nella forma attr1=value1 attr2=value2...; le azioni sono eseguite nella fase di processing, e danno origine a codice Java specifico per la loro esecuzione.
- ***Scripting element:*** sono porzioni di codice in un scripting language specifico nelle direttive; il linguaggio di default e' lo stesso java specifico per la loro esecuzione.
- ***Scriplet:*** con sintassi <%code%>; sono porzioni di codice nello scripting language che danno origine a porzioni di codice java; generalmente inserite nel metodo service() del Servlet; se contengono dichiarazioni di variabili queste saranno variabili locali valide solo nell'ambito di una singola esecuzione del Servlet; se il codice contiene istruzioni che scrivono sullo stream di output, il contenuto mandato in output sarà inserito nella pagina di risposta nella stessa posizione in cui si trova lo scriplet.
- ***Declaration:*** con sintassi <%! Declaration [declaration]...%>; sono dichiarazioni che inserite nel Servlet come elementi di classe, al di fuori di qualunque metodo; possono essere sia variabili di classe che metodi. Se si tratta di variabili, la loro durata e' quella del Servlet stesso; di conseguenza sopravvivono e conservano il loro valore nel corso di tutte le esecuzioni dello stesso oggetto Servlet.
- ***Expression:*** con sintassi <%=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'espressone JSP.

2.11 Oggetti espliciti

Si e' visto come esiste una corrispondenza uno a uno fra una pagina JSP e il Servlet nel quale viene tradotta. Inoltre l'esecuzione della pagina JSP corrisponde all'esecuzione del metodo service() del servlet. Come e' noto, il servlet manipola una serie di oggetti per svolgere il suo lavoro, principalmente

i due oggetti ServletRequest e ServletResponse, su cui si basa tutto il meccanismo di funzionamento. Quindi per poter usufruire dei servizi dei servlet nella pagina JSP, occorre avere un modo per accedere a questi oggetti: a questo scopo esistono gli oggetti impliciti JSP, utilizzabili in tutti gli scriplet.

Diamone una lista aggiungendo qualche breve nota per ciascuno di essi:

- ***Request:*** corrisponde generalmente all'oggetto ServletRequest passato come parametro al metodo Service() del servlet; può essere una qualunque sottoclasse di Servlet Request; generalmente si tratta di una sottoclasse di Httpservletrequest.
- ***Response:*** corrisponde all'oggetto ServletResponse passato come parametro al metodo service() del Servlet; può essere una qualunque sottoclasse di ServletResponse; generalmente si tratta di una sottoclasse di HttpServlte Response.
- ***Out:*** perché il container deve fornire un meccanismo di buffering, non e' dato accesso direttamente all'oggetto PrintWriter restituito da response.getWriter(). L'oggetto out e' invece uno stream bufferizzato di tipo javax.servlet.jsp.JspWriter, restituito da un metodo del PageContext. Il trasferimento sull'output stream del ServletResponse avviene in un secondo tempo, dopo che tutti i dati sono stati scritti sull'oggetto out.
- ***Page:*** e' un riferimento all'oggetto che gestisce la richiesta corrente, che si e' vista essere generalmente un servlet. In java corrisponde al this dell'oggetto, pertanto e' di scarsa utilità. Si presume risulti utile con altri linguaggi script.
- ***PageContext:*** si tratta di un oggetto della classe javax.servlet.jsp.PageContext utilizzata prevalentemente per incapsulare oggetti e features particolari di ciascuna implementazione del container.
- ***PageContext*** contiene ad esempio un metodo che restituisce l'oggetto out, altri che restituiscono gli oggetti session, application, e così via.
- ***Session:*** corrisponde all'oggetto HttpSession del servlet, viene restituito da un metodo del pageContext.
- ***Application:*** corrisponde al ServletContext del servlet, viene restituito da un metodo del PageContext.
- ***Config:*** corrisponde al ServletConfig del servlet, viene restituito da un metodo del PageContext.

- ***Exception:*** questo oggetto e' disponibile solo all'interno di una pagina di errore.

Capitolo 3
Analisi dell'applicazione web con UML

Volendo fornire un'adeguata documentazione relativa ad un'applicazione web che diventerà parte integrante del sito della facoltà di ingegneria, si è deciso di utilizzare l'UML. Quest'ultimo è un tipo di linguaggio di progettazione universale in grado di rappresentare sistemi molto diversi senza differenze legate alla tecnologia. Questo linguaggio grazie alla sua versatilità ed immediatezza è quindi in grado di mostrare quali sono le caratteristiche di base e le funzionalità principali di questa applicazione.

L'UML per sua stessa natura è strutturato in varie parti che, osservando il progetto da diversi punti di vista, permettono di evidenziare i molti aspetti che caratterizzano un'applicazione.

Vedremo nei prossimi paragrafi quali siano queste parti mettendo di volta in volta in evidenza quali siano i tratti principali di ciascuna di esse.

3.1 Introduzione

Si vuole progettare un software in grado di gestire alcuni aspetti legati all'iscrizione agli esami e alla visualizzazione delle date degli appelli. Questo applicazione mira a soddisfare principalmente le esigenze di due tipi principali di utenti che sono gli studenti ed i docenti. Questo non vuol dire che questa applicazione è preclusa a utenti privi di account, ma soltanto che le maggiori funzionalità sono messe a disposizione dei soli utenti registrati. Questo fatto è stato previsto per poter garantire già dall'inizio un certo livello di sicurezza. È bene sottolineare che comunque quest'applicazione consente la possibilità, per i soli studenti, di poter ottenere, in tempo reale, un nuovo account, di modo che, se si desidera ottenere maggiori informazioni o poter compiere operazioni di più alto livello, è sufficiente solo registrarsi. Un ulteriore aspetto che riguarda l'utenza ed in particolar modo la sicurezza è quello legato al fatto che ciascun utente non ha la possibilità di eseguire tutte le operazioni che desidera ma soltanto un numero ristretto di esse.

Da quanto enunciato precedentemente si ricava che i tipi di utenti che possono accedere a quest'applicazione sono:

- ***Studente:*** sono coloro che sono regolarmente iscritti alla facoltà di ingegneria e che posseggono un adeguato account (username, password).
- ***Docente:*** sono coloro che detengono il corso di uno o più insegnamenti all'interno della facoltà di ingegneria e che posseggono un adeguato account (username, password).
- ***Utente generico:*** sono tutti coloro che non sono possessori di un adeguato account (username, password).

Prendiamo ora in considerazione quali sono le funzionalità messe a disposizione classificandole in base agli utenti che possono utilizzarle.

Studente:

- visualizzazione delle informazioni relative alle date di appelli disponibili;
- visualizzazione del numero degli studenti iscritti a ciascun appello disponibile;
- iscrizione ad un appello o ritiro da esso;
- visualizzazione dei voti conseguiti in prove d'esame precedentemente sostenute.

Docente:

- visualizzazione delle informazioni relative alle date di appelli disponibili;
- visualizzazione del numero degli studenti iscritti a ciascun appello disponibile;
- modifica della data di chiusura della lista d'iscrizione ad un esame;
- visualizzazione degli studenti iscritti ai propri esami

- eliminazione dalla lista di iscrizione di tutti quegli studenti che, pur essendosi iscritti, non hanno sostenuto la prova;
- inserimento dei voti conseguiti dagli studenti in una prova d'esame;
- pubblicazione dei voti sul web in modo che siano messi a disposizione degli studenti.

Utente generico:

- ricerca delle date degli appelli disponibili sulla base di alcuni criteri di selezione e visualizzazione dei risultati ottenuti;
- possibilità di registrazione nel database in modo da poter ottenere un account.

3.2 Requisiti funzionali

REQUISITO #1:

Operazione di visualizzazione delle date degli appelli per mezzo di un sistema di ricerca messo a disposizione dall'applicazione.

INPUT:

Nome dell'insegnamento e/o cognome del docente e/o sessione e/o corso di studi di cui si vuole avere le informazioni.

PROCESSING:

Il sistema sulla base dei dati immessi seleziona le date disponibili.

OUTPUT:

Schermata in cui compaiono le date attualmente disponibili.

REQUISITO #2:

Operazione di visualizzazione delle date disponibili attualmente.

INPUT:

Immissione del nome dell'insegnamento e della sessione della quale si vuole visualizzare le date.

PROCESSING:

Il sistema seleziona le date che corrispondono ai dati immessi.

OUTPUT:

In caso ci siano date disponibili vengono rese visibili, in caso contrario viene visualizzato un messaggio in cui si sottolinea l'assenza di date disponibili.

REQUISITO #3:

Operazione di registrazione di un utente generico.

INPUT:

Immissione del cognome, nome, numero di matricola, corso di studi dello studente e username e password che si vogliono utilizzare.

PROCESSING:

Il sistema verifica che lo studente non si sia già registrato, in caso contrario procede alla sua registrazione.

OUTPUT:

In caso tutto avvenga regolarmente visualizzazione di un messaggio in cui si sottolinea il successo dell'operazione appena conclusa, in caso contrario messaggio di errore.

REQUISITO #4:

Operazione di visualizzazione del numero degli studenti iscritti ad un appello.

INPUT:

Data dell'appello e nome dell'insegnamento di cui si vuole conoscere l'informazione.

PROCESSING:

Il sistema sulla base dei dati immessi conta il numero degli studenti iscritti a quell'insegnamento.

OUTPUT:

Schermata in cui compare l'effettivo numero degli studenti iscritti a quell'appello.

REQUISITO #5:

Operazione di iscrizione/cancellazione di uno studente ad un esame.

INPUT:

Nome dell'insegnamento e data dell'appello al quale ci si vuole iscrivere o dal quale ci si vuole ritirare.

PROCESSING:

Il sistema procede all'esecuzione dell'operazione richiesta dall'utente preoccupandosi di verificare che non sia già avvenuta precedentemente.

OUTPUT:

In caso di ripetizione dell'operazione richiesta viene visualizzato un messaggio di errore, in caso tutto sia corretto viene visualizzato un messaggio di conferma della corretta esecuzione della operazione.

REQUISITO #6:

Operazione di visualizzazione dei voti conseguiti in una prova d'esame precedentemente sostenuta da uno studente.

INPUT:

Immissione della data e dell'insegnamento di cui si visualizzare il proprio voto.

PROCESSING:

Il sistema sulla base delle informazioni immesse verifica che ci siano o meno voti da visualizzare

OUTPUT:

Se ci sono voti si assiste alla visualizzazione del voto in caso contrario viene visualizzato un messaggio in cui si sottolinea che non sono disponibili voti relativi a quello studente.

REQUISITO #7:

Operazione di modifica della data di chiusura della lista di iscrizione di un appello.

INPUT:

Nome dell'insegnamento ,data dell'appello e numero dei giorni a partire da quali la lista viene chiusa.

PROCESSING:

Il sistema controlla l'esattezza del valore immesso e se corretto lo immette.

OUTPUT:

Se il valore immesso è corretto messaggio di conferma dell'avvenuta operazione in caso contrario messaggio di errore.

REQUISITO #8:

Operazione di visualizzazione degli studenti iscritti ad un determinato esame.

INPUT:

Nome dell'insegnamento e data dell'appello di cui si vuole visualizzare gli iscritti

PROCESSING:

Il sistema sulla base dei dati immessi ricava i dati degli studenti eventualmente iscritti.

OUTPUT:

In caso ci siano studenti iscritti visualizza i loro dati, in caso contrario sottolinea l'assenza di questi ultimi.

REQUISITO #9:

Operazione di cancellazione di uno studente da una lista di esame.

INPUT:

Nome dell'insegnamento, data dell'appello e identificatore dello studente.

PROCESSING:

Il sistema verifica che l'identificatore immesso corrisponda a quelli messi a disposizione e in caso affermativo procede alla cancellazione di quello studente.

OUTPUT:

Nel caso il valore immesso sia corretto visualizzazione di un messaggio in cui si mostra che l'operazione è avvenuta con successo, messaggio di errore in caso contrario.

REQUISITO #10:

Operazione di immissione dei voti degli studenti.

INPUT:

Immissione del nome dell'insegnamento,data dell'appello, dell'identificatore dello studente e voti da assegnare.

PROCESSING:

Il sistema verifica che l'identificatore immesso corrisponda a quelli messi a disposizione e in caso affermativo procede all'immissione dei voti di quello studente.

OUTPUT:

In caso tutto avvenga regolarmente visualizzazione dei voti degli studenti aggiornata, in caso contrario messaggio di errore.

REQUISITO #11:

Operazione di immissione in rete dei voti conseguiti dagli studenti.

INPUT:

Immissione del nome dell'insegnamento, data dell'appello.

PROCESSING:

Il sistema mette in rete la lista che diviene cosí disponibile per la consultazione degli studenti.

OUTPUT:

In caso tutto avvenga regolarmente visualizzazione della data fra quelle attualmente in rete.

3.3 Use cases

3.3.1 Diagramma degli attori

Questo diagramma mostra i tipi di utenti che sono ammessi nel sistema informativo. Chiunque può accedere come “utente generico” ma, come si avrà modo di constatare, i suoi diritti sono veramente limitati. Appartengono a questa categoria tutti coloro che si collegano al sito internet della facoltà. Gli “utenti registrati” ereditano tutte le caratteristiche dei precedenti e le ampliano a seconda del ruolo che ricoprono.

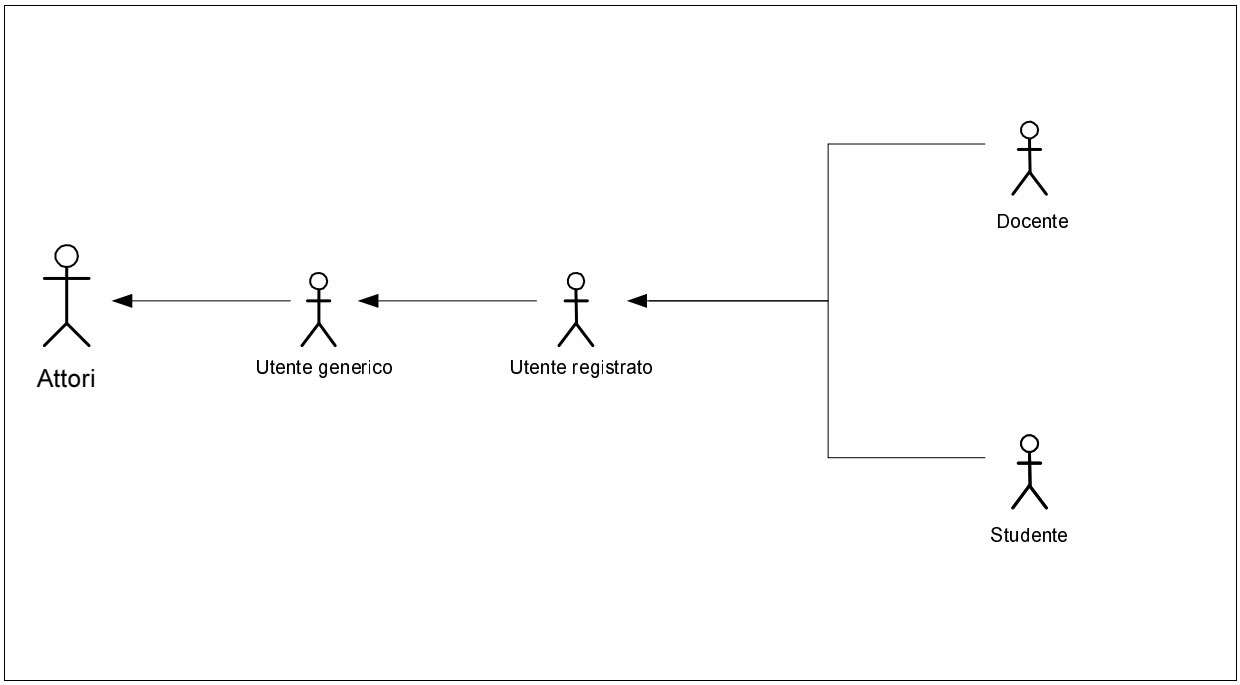


Figura III.1: Diagramma degli attori

3.3.2 Diagramma delle interfacce

Ad ognuna di queste categorie corrisponde un'interfaccia specifica, cioè ogni utente vede il sistema in un modo differente. Questo risulta essere opportuno per diversi motivi:

- non tutti gli utenti hanno le medesime conoscenze quindi bisogna stare attenti all'integrità dei dati;
- gli utenti non registrati non possono accedere ad una parte dei dati;
- alcune funzionalità riguardano solamente utenti particolari.

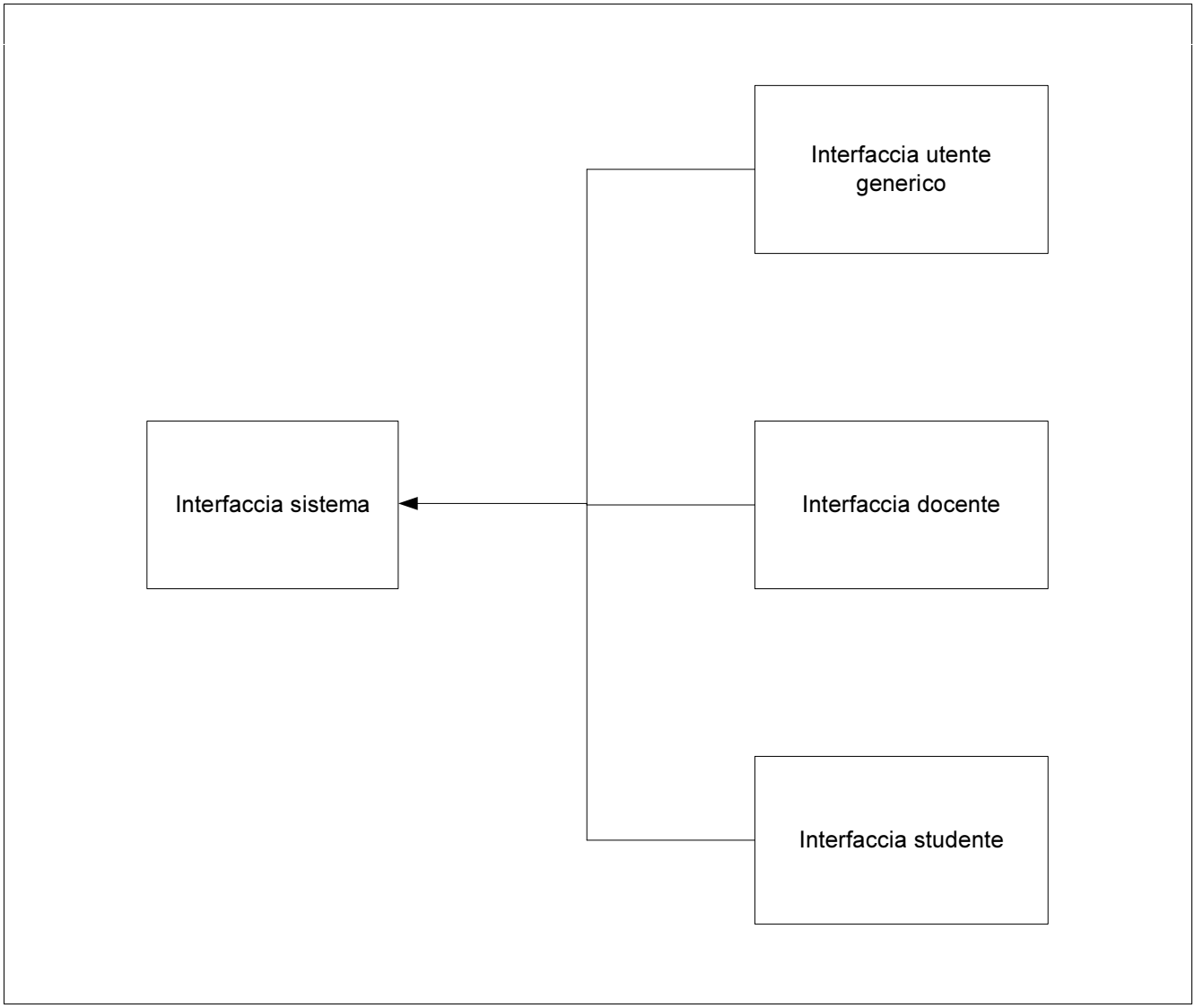


Figura III.2: Diagramma delle interfacce

3.3.3 Diagramma dei casi d'uso

Mostriamo ora i principali usi dell'applicativo in esame. Questi diagrammi mettono in relazione gli utenti con le principali parti del sistema che si andrà a modellare. Per ogni categoria di utente

vengono specificate le funzionalità di cui possono disporre ma non viene detto nulla su come verranno realizzate.

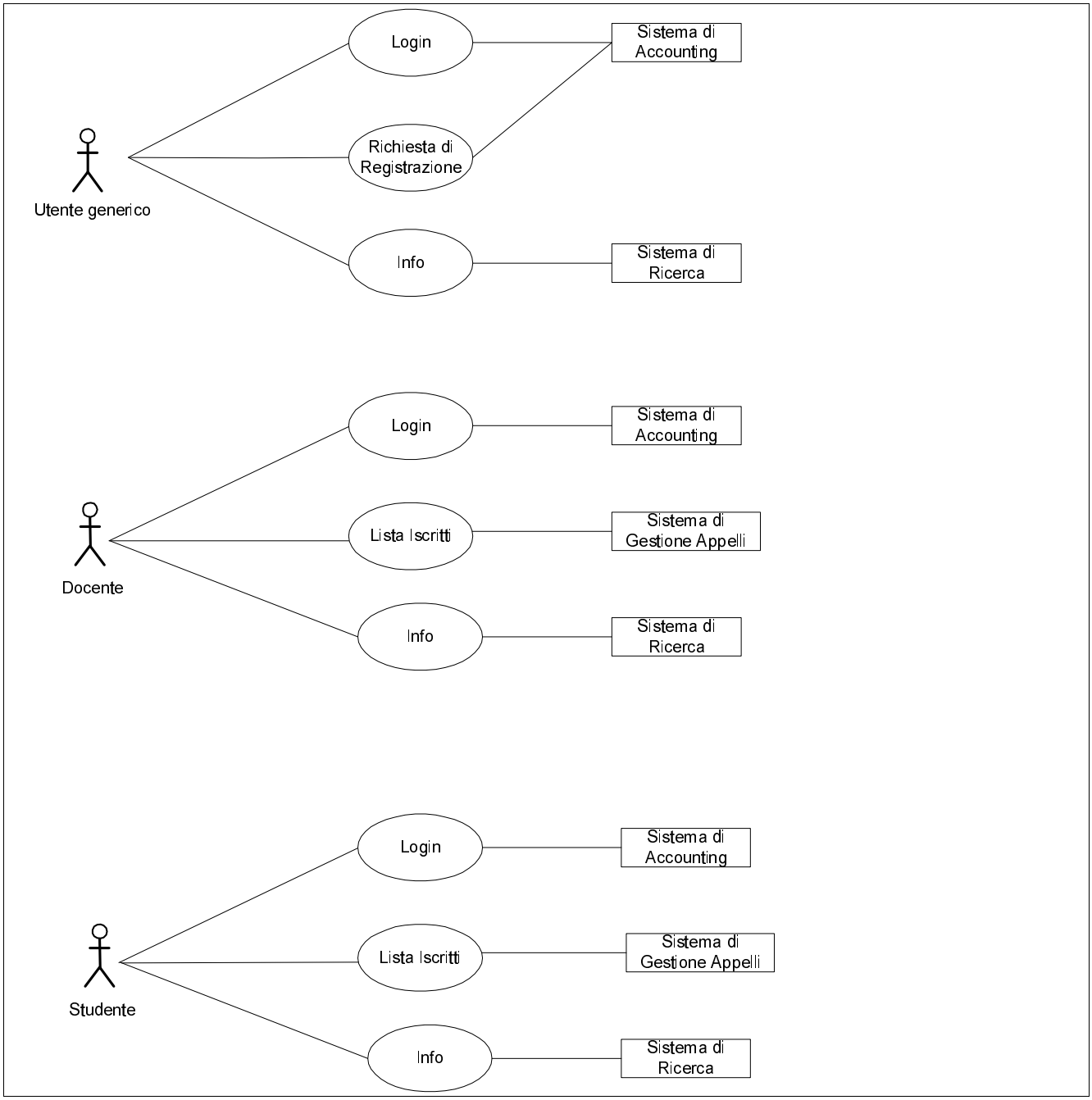
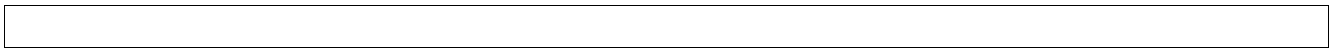


Figura III.3: Diagramma dei casi d'uso

3.4 Class Diagram

Sono diagrammi di tipo statico per modellare le classi che verranno create nel progetto e riguardano la struttura e le relazioni che intercorrono fra di esse.
Il primo mostra la classificazione degli utenti del sistema. “Utente Generico” è chiunque si collega prima di aver eseguito il login. Questa classe di utenti quindi non ha attributi di identificazione e nemmeno metodi (sono sconosciuti ed anonimi quindi è impossibile agire su di essi). Possiedono diritti molto limitati e per operazioni “standard” che non possono recare danno alcuno al sistema. Se già registrati, eseguendo un’operazione di login vengono riconosciuti come tali e vedono ampliarsi le proprie possibilità d’azione.



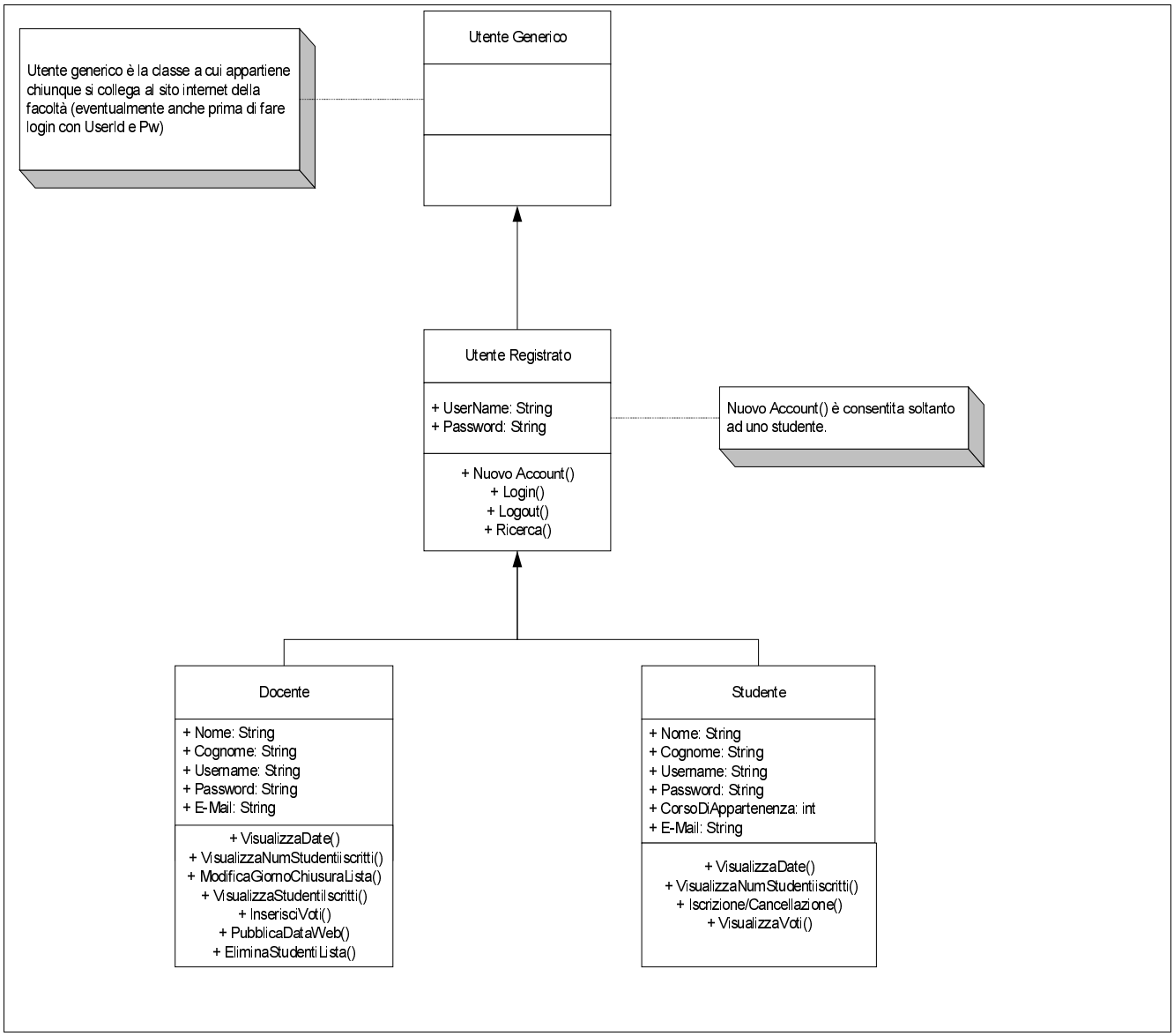


Figura III.4: Class Diagram dei diversi tipi di utenti

Il secondo mostra le operazioni consentite ad utente “Generico”.

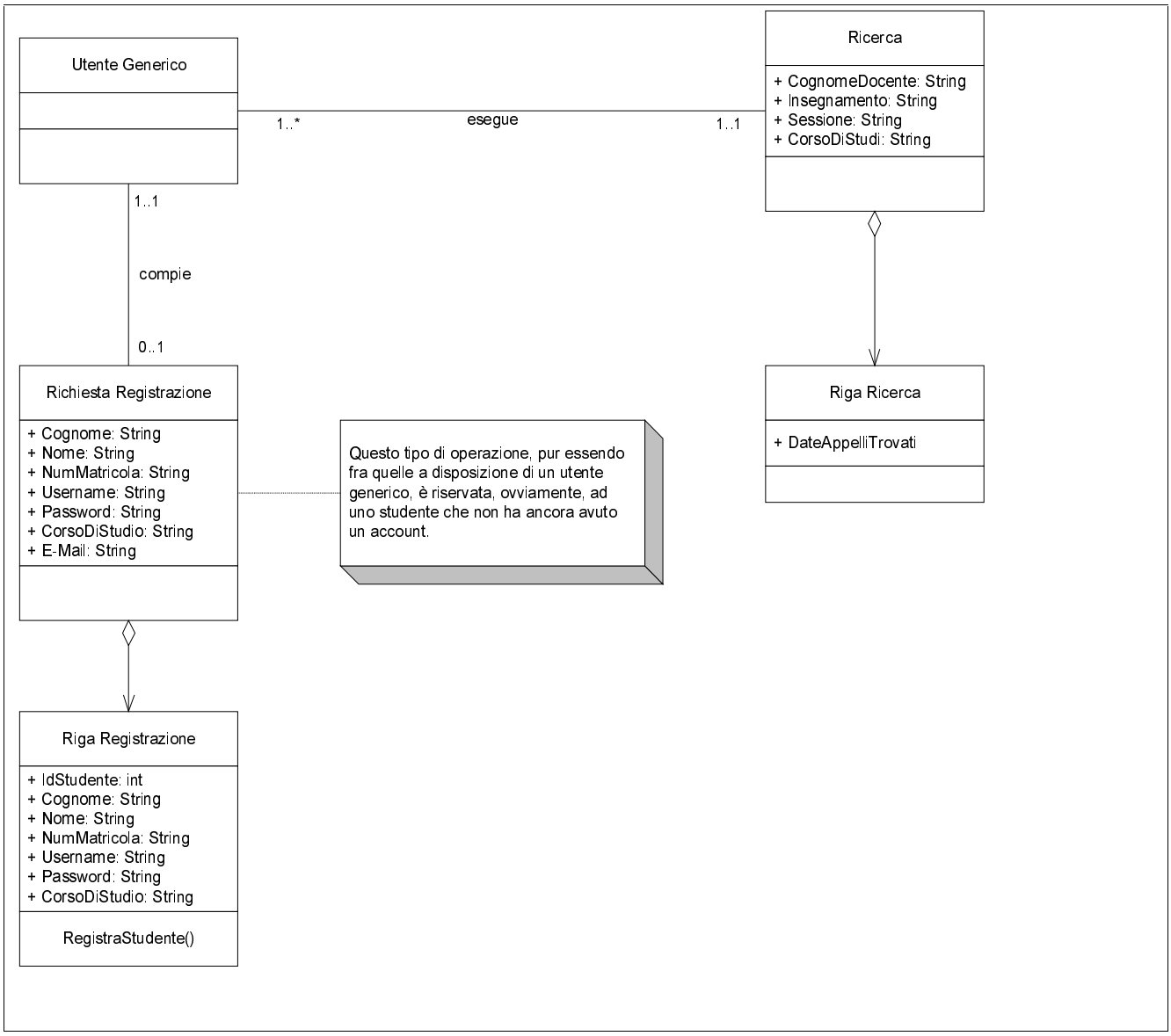


Figura III.5: Class Diagram dell'utente generico

Il terzo rappresenta le operazioni consentite ad uno studente.

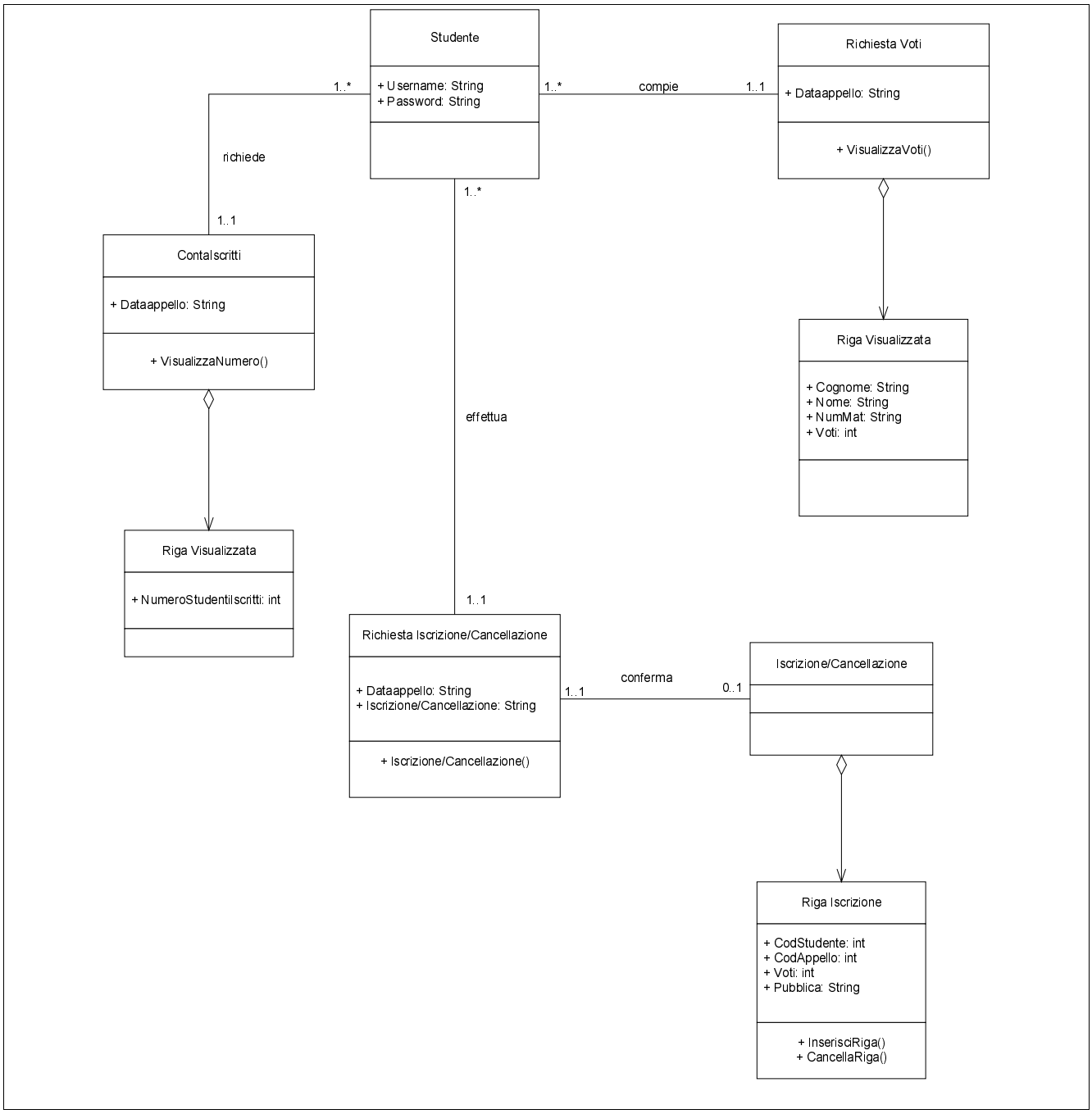


Figura III.6: Class Diagram dello studente

Il quarto ed ultimo diagramma rappresenta le operazioni consentite ad un docente.

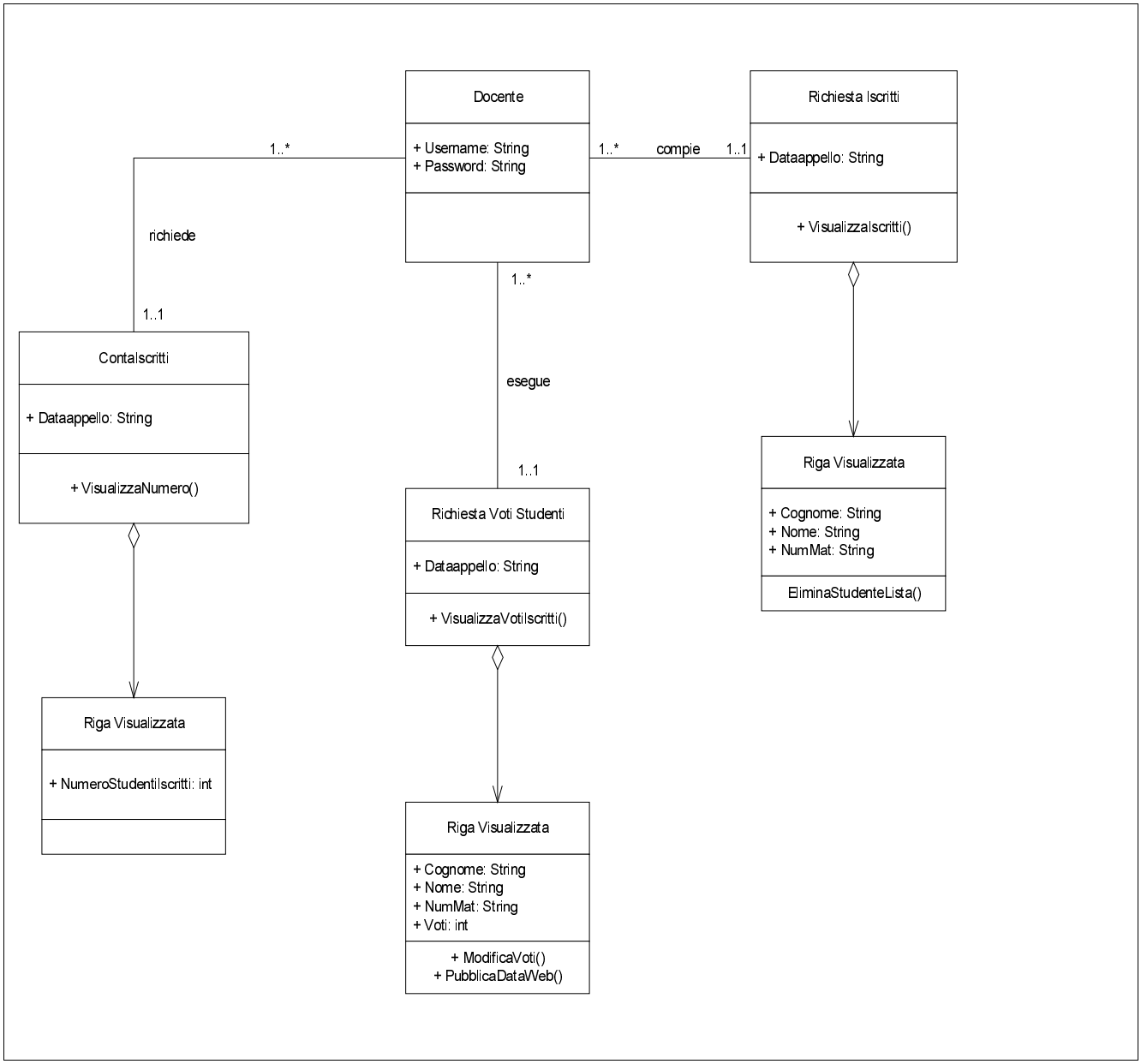


Figura III.7: Class diagram del docente

3.5 Sequence Diagram

Vengono ora riportati alcuni diagrammi delle sequenze che permettono di comprendere la dinamica di esecuzione delle varie operazioni.

3.5.1 Registrazione di un Utente Generico

Dopo l'accesso al portale e dopo aver scelto il comando di registrazione all'utente viene mostrata una form da compilare (con tutti i dati del caso). Il sistema che gestisce gli account deve solo controllare che non vi siano dati duplicati e crearne uno nuovo. Nel caso in cui un utente provi a registrarsi più volte occorre rifiutare la registrazione.

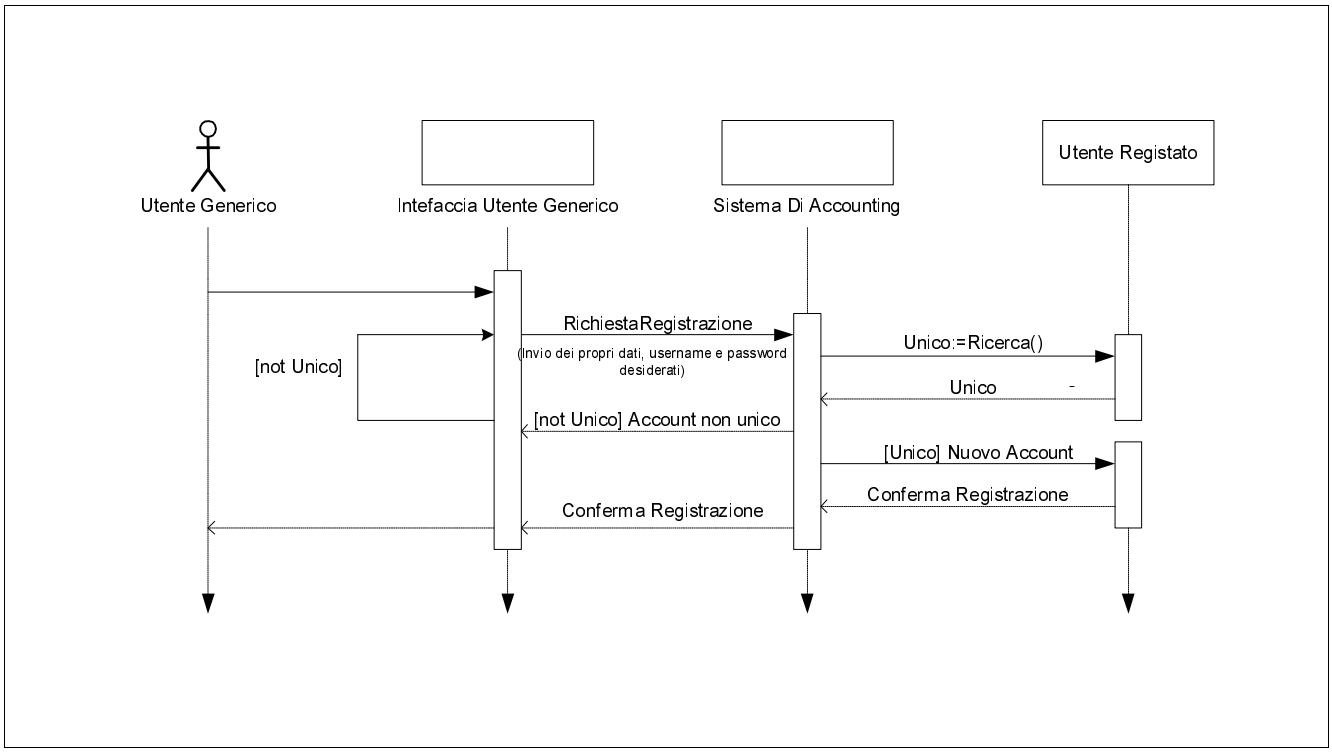


Figura III.8: Sequence diagram della registrazione di un utente generico

3.5.2 Login e Logout degli utenti con account

Il login serve per far accedere al sistema solo quegli utenti che sono registrati. Dopo questa operazione possono essere fatte tutte le operazioni che si desidera (naturalmente fra quelle che si è abilitati ad eseguire). Al termine della sessione di lavoro si fa il logout per disconnettere l'utente dal sistema.

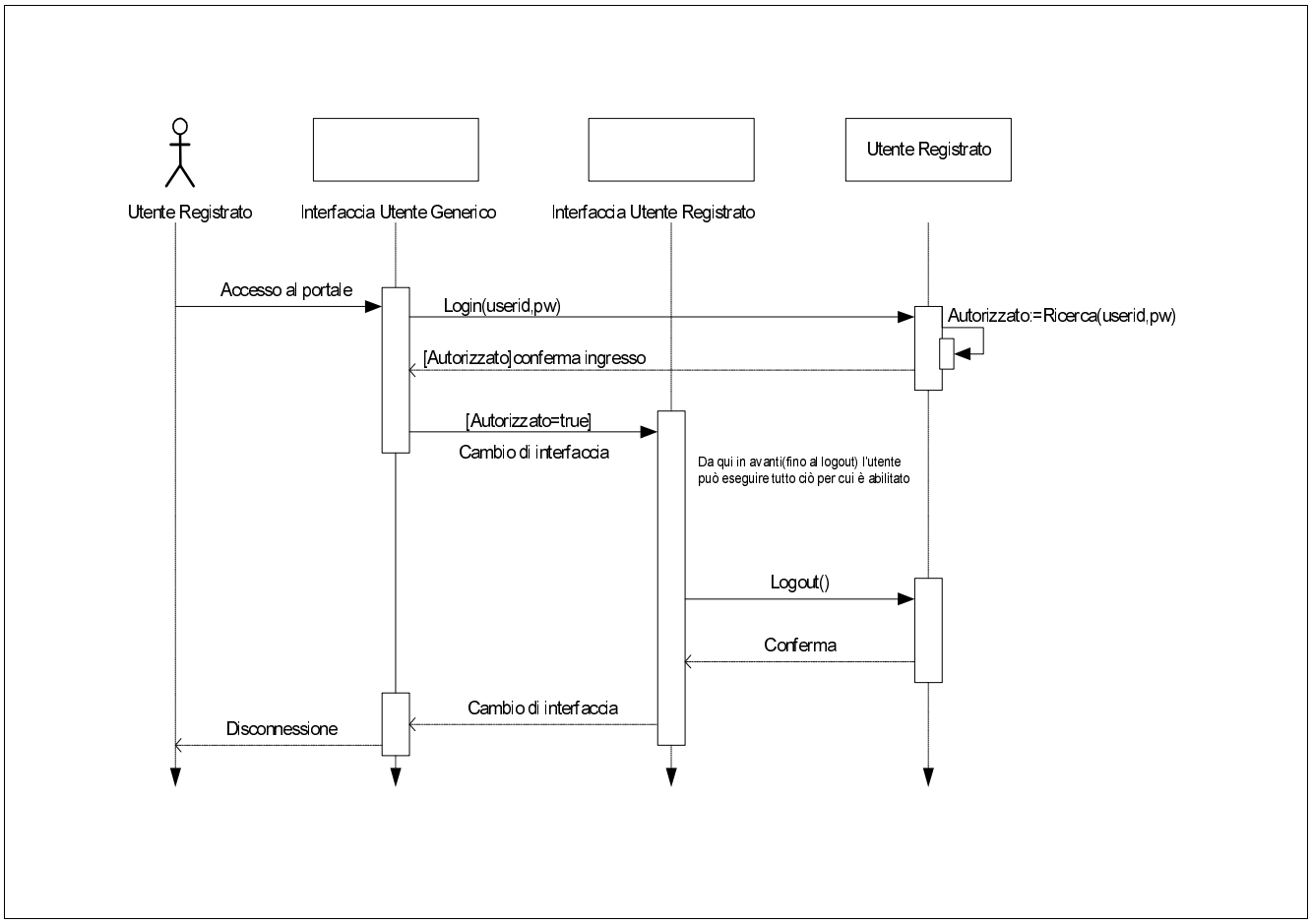


Figura III.9: Sequence diagram del login e logout di un utente con account

3.5.3 Iscrizione/Cancellazione di uno studente

Operazione grazie alla quale è consentito ad uno studente la possibilità di iscriversi o cancellarsi da una lista d'esame. Lo studente al momento dell'esecuzione dell'operazione dovrà specificare i propri dati. Il sistema prima di eseguire l'operazione eseguirà un controllo: esegue la verifica della non ripetizione dell'operazione appena richiesta.

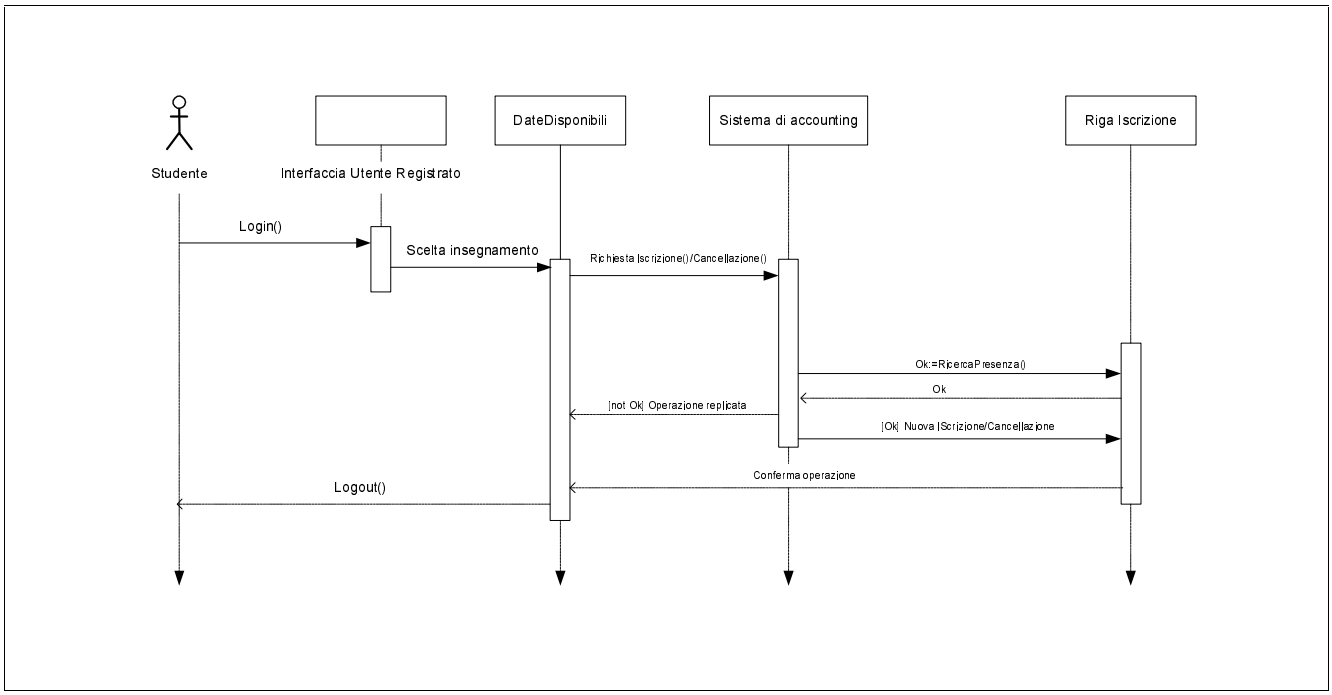


Figura III.10: State diagram dell'iscrizione/cancellazione di uno studente ad/da una lista d'esame

3.5.4 Ricerca avanzata da parte di un Utente Generico

L'Utente Generico ha la possibilità, inserendo uno o più dati, di visualizzare le date degli appelli attualmente disponibili.

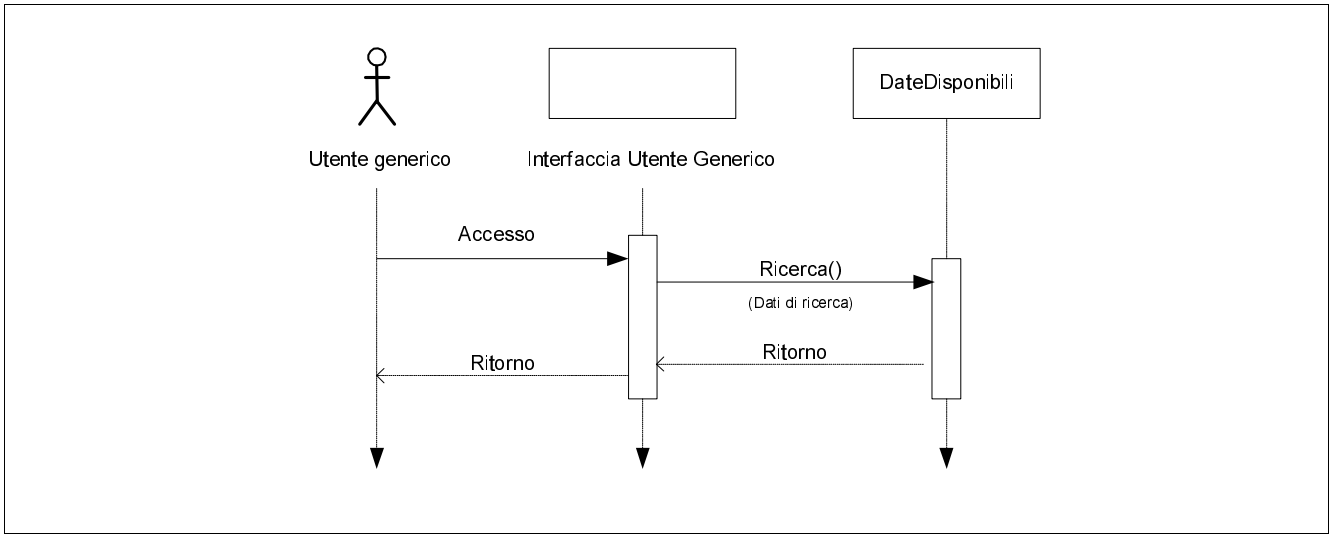


Figura III.11: Sequence diagram della ricerca avanzata svolta da un utente generico

3.5.5 Operazioni che può eseguire un docente su una lista d'esame

Questo diagramma mostra quali sono le operazioni che può compiere un docente sulla lista degli studenti iscritti una volta che si è chiusa. Il docente può eliminare dalla lista tutti quegli studenti che pur essendosi iscritti non hanno sostenuto la prova, assegnare a ciascuno di essi il voto che gli compete ed infine pubblicare questa lista in rete permettendo agli studenti di consultarla.

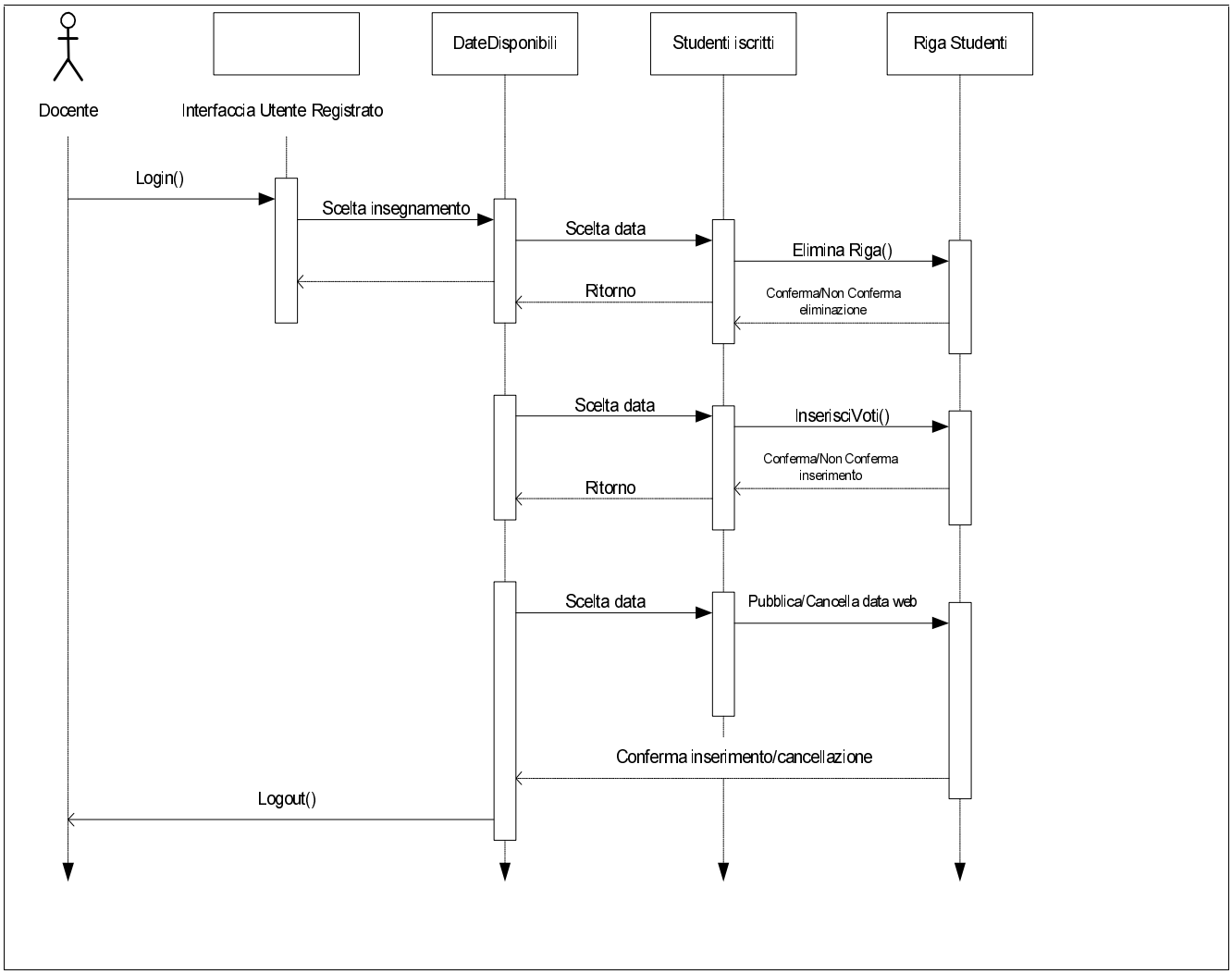


Figura III.12: Sequence diagram delle operazioni eseguibili da un docente su una lista d'esame

3.6 Action Diagram

I diagrammi delle attività rappresentano una serie di stati d'azione (cioè che esprimono un'azione).

Il primo mostra come avviene la ricerca dinamica richiesta da un utente generico.

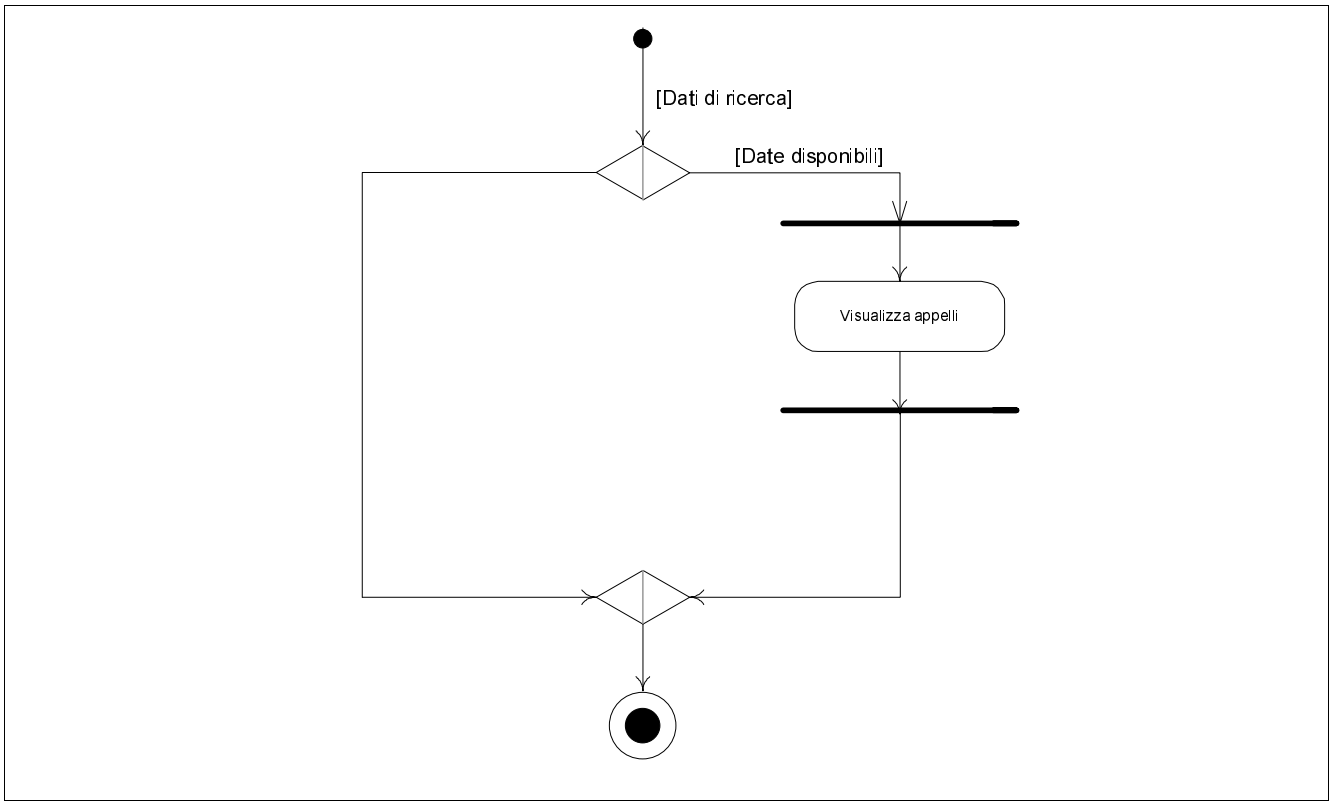


Figura III.13: Action diagram di una ricerca dinamica

Il secondo mostra le operazioni che può compiere un docente su una lista d'esame.

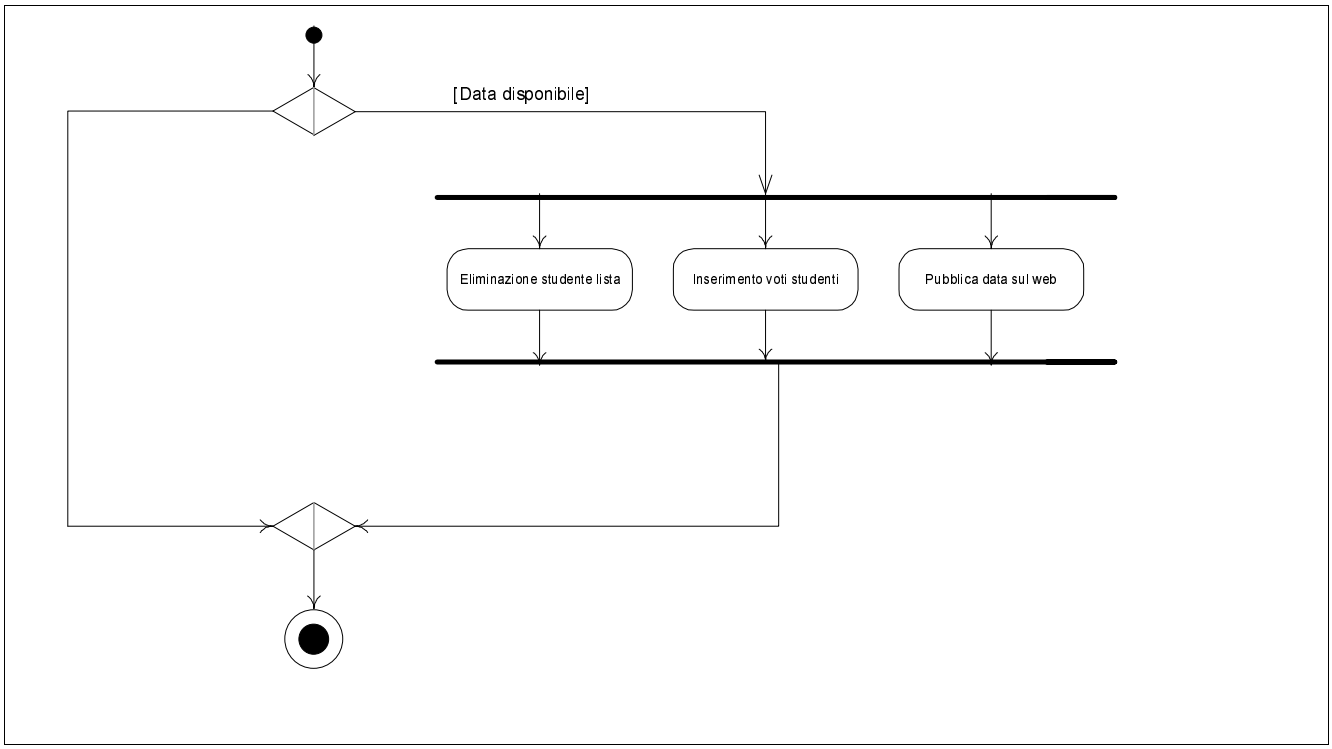


Figura III.14: Action diagram delle operazioni eseguite da un docente su una lista d'esame

Il terzo ed ultimo mostra le operazioni che può compiere sulla lista d'esame. Lo studente può iscriversi o cancellarsi e visualizzare i voti della prova appena sostenuta appena il docente la metta a disposizione.

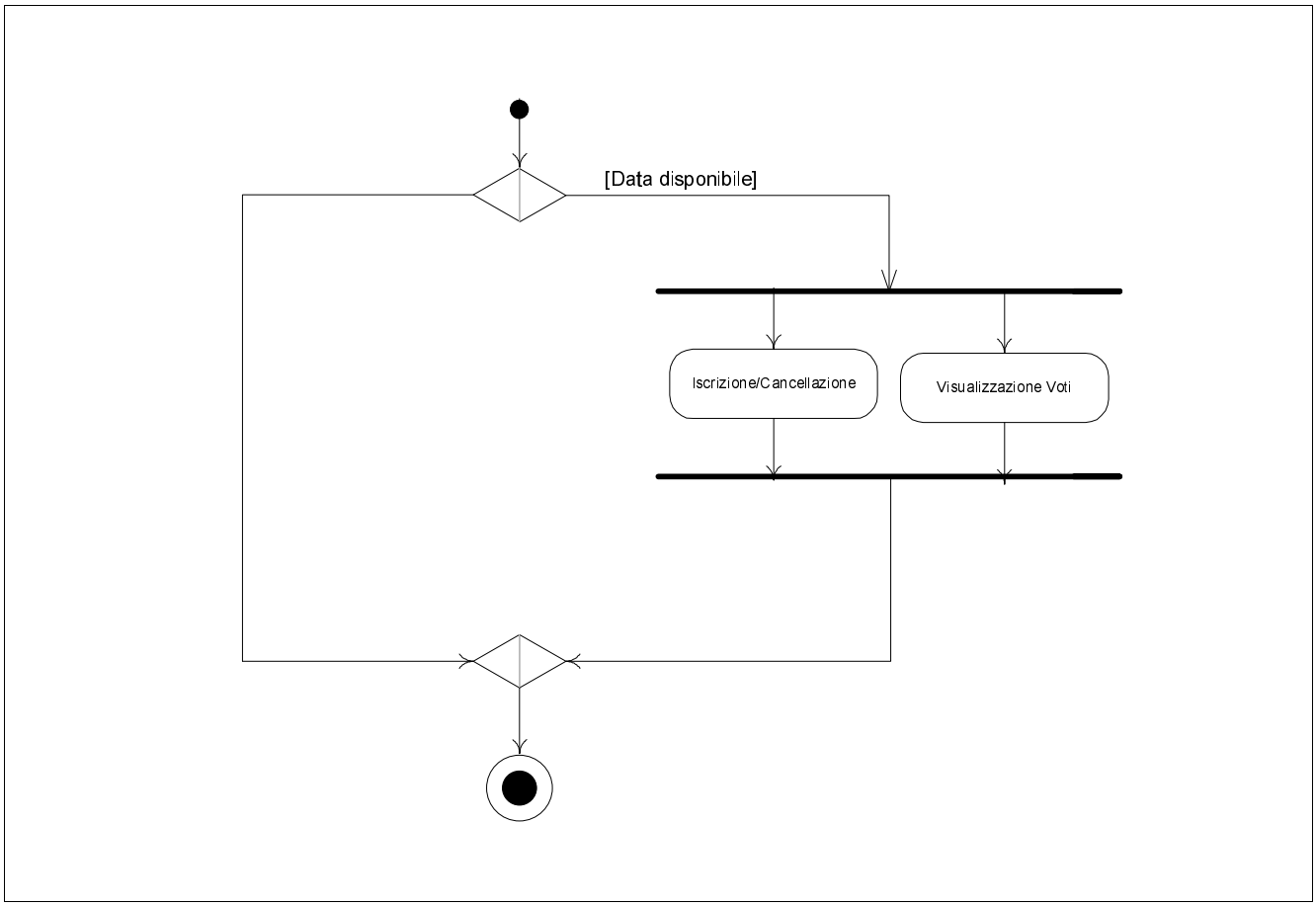


Figura III.15: Action diagram delle operazioni che può compiere uno studente su una lista d'esame

3.7 State Diagram

Servono per generalizzare quanto detto in precedenza: rappresentano i possibili stati per i quali un oggetto o un'interazione può passare.

UTENTE GENERICO

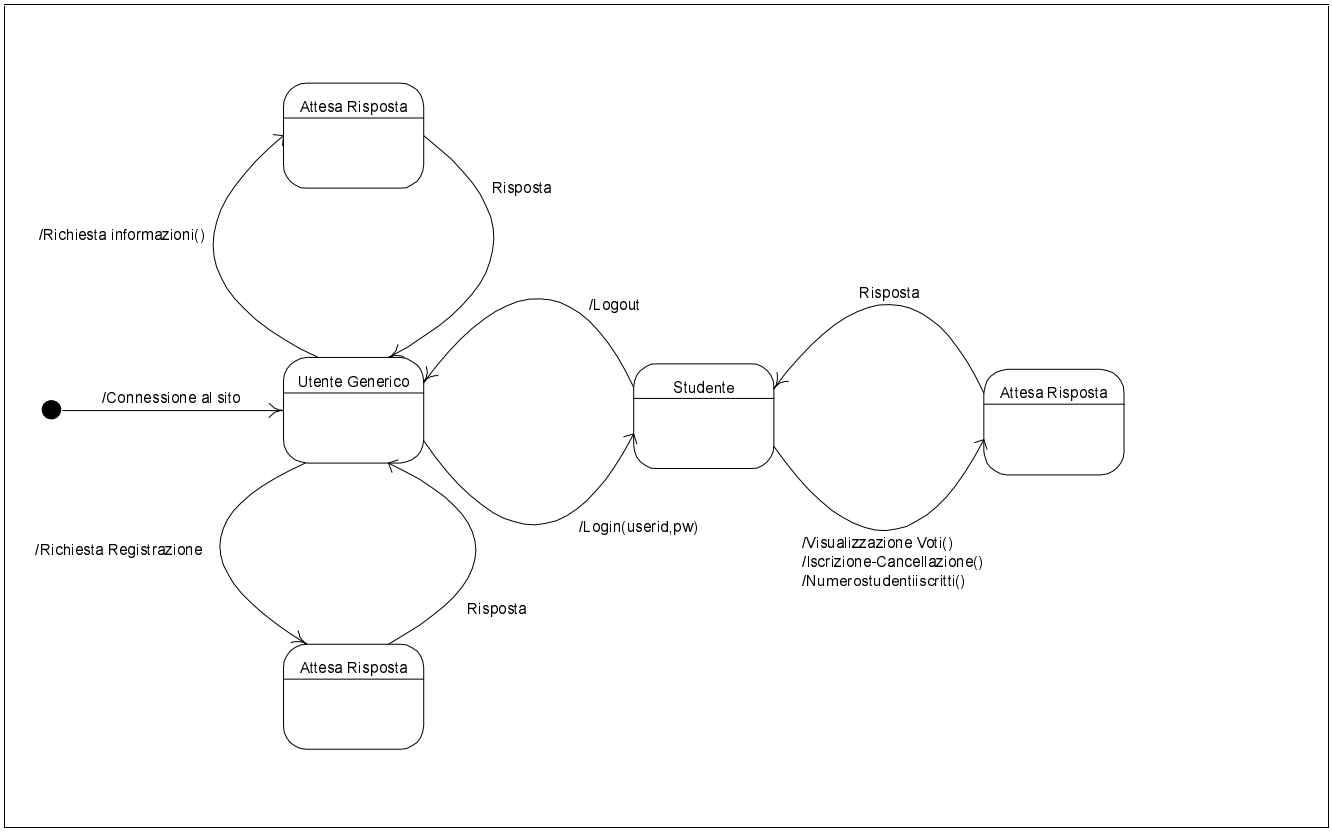


Figura III.16: State diagram delle operazioni che può compiere un utente generico

STUDENTE

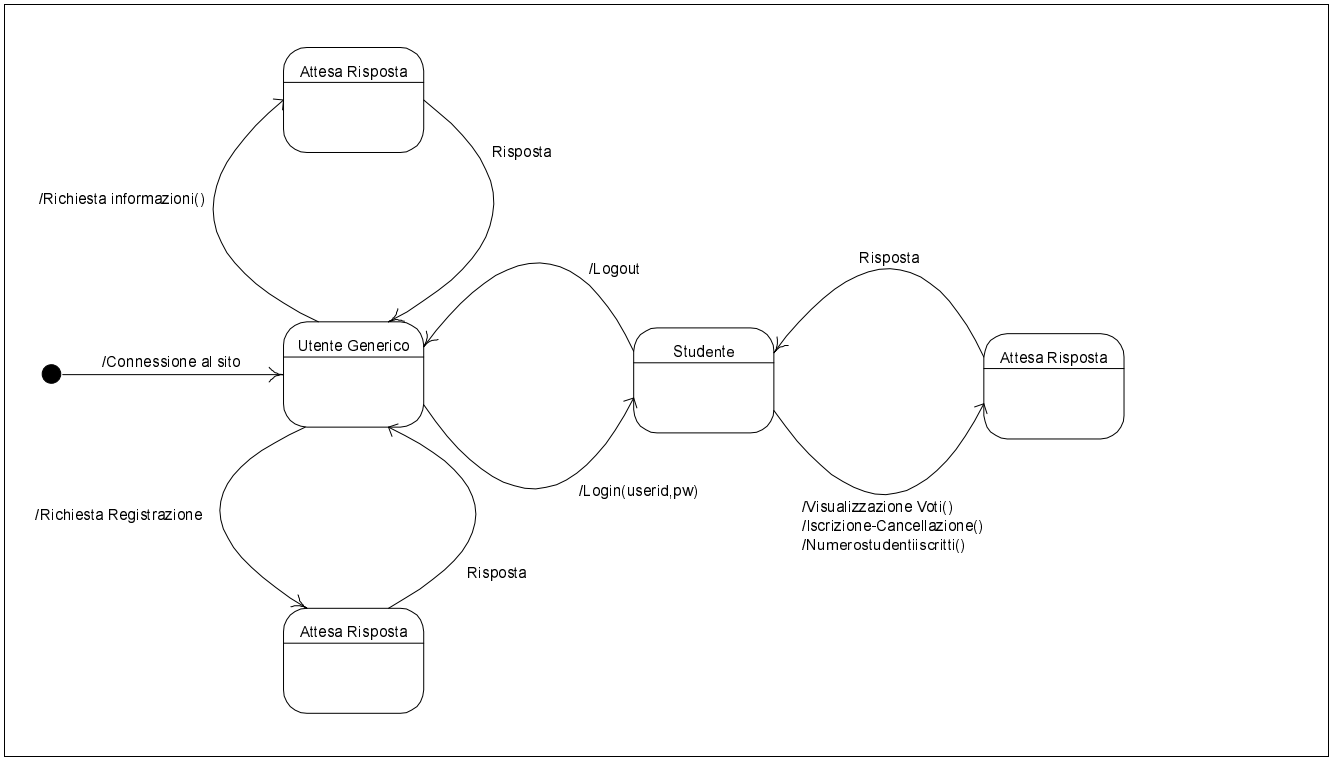


Figura III.17: State diagram delle operazioni che può compiere uno studente

DOCENTE

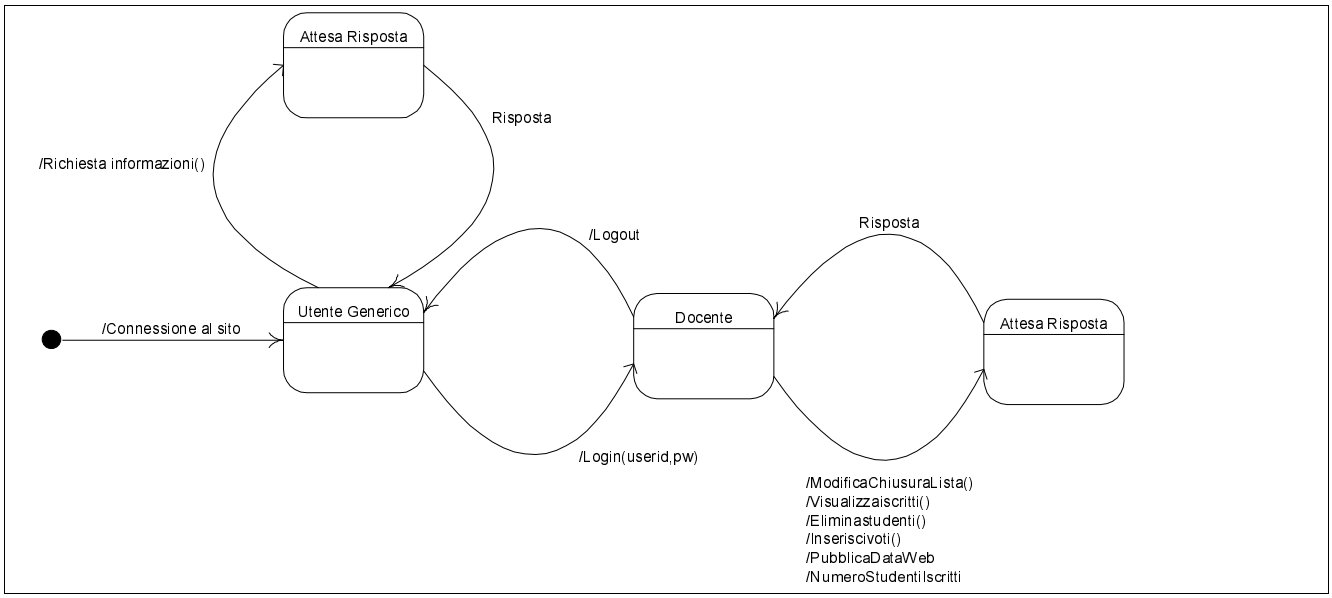


Figura III.18: State diagram delle operazioni che può compiere un docente

3.8 DFD (Data Flow Diagram)

Il DFD è uno strumento previsto da OMT ma non da UML. Tuttavia si è deciso di utilizzarlo per dare un'idea della scomposizione funzionale di un processo (che quindi diviene un concetto affinabile per stadi).

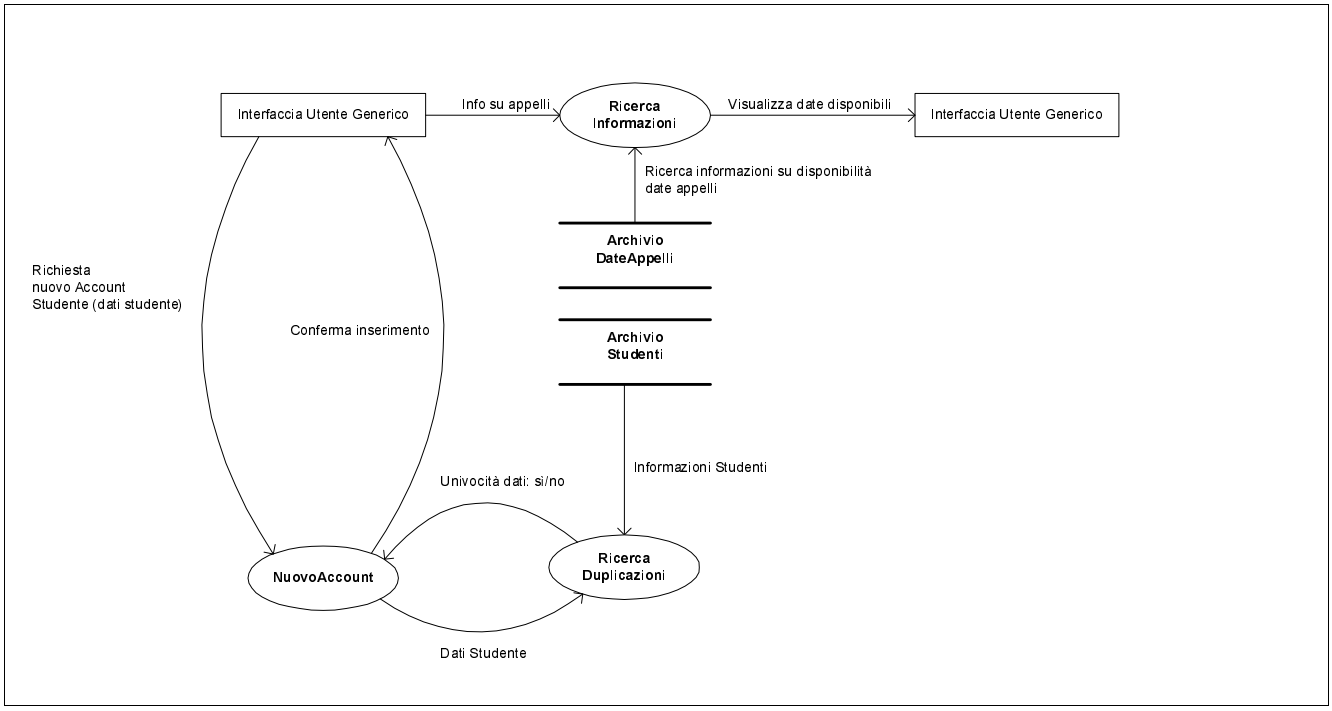


Figura III.19: DFD dell'utente generico

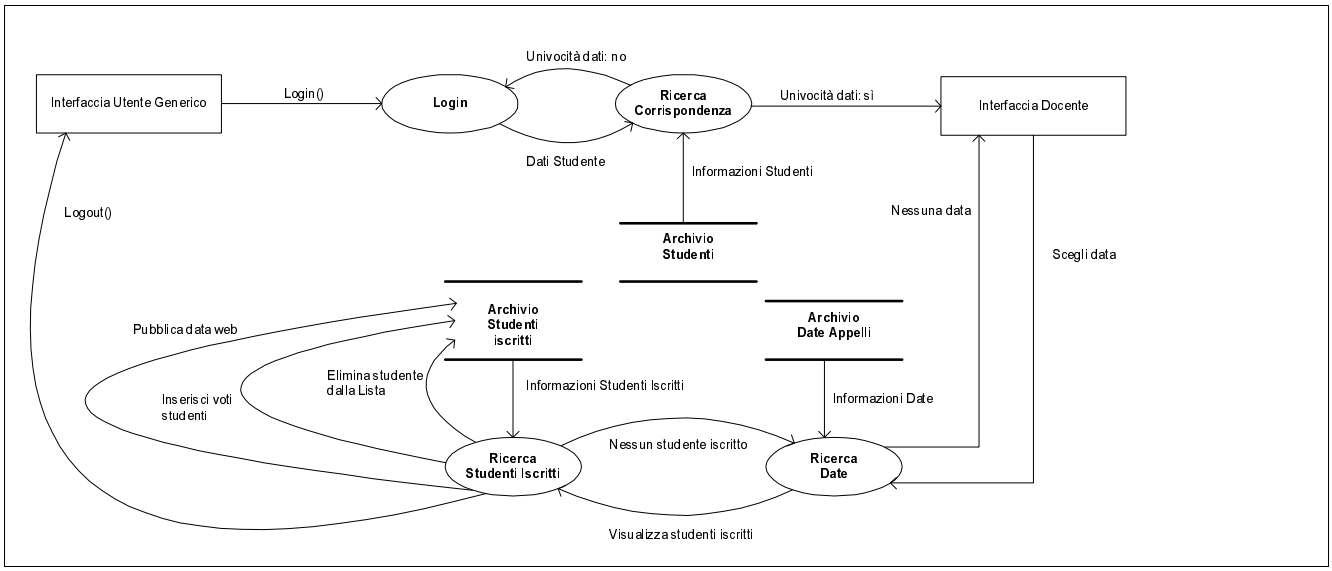


Figura III.20: DFD di un docente

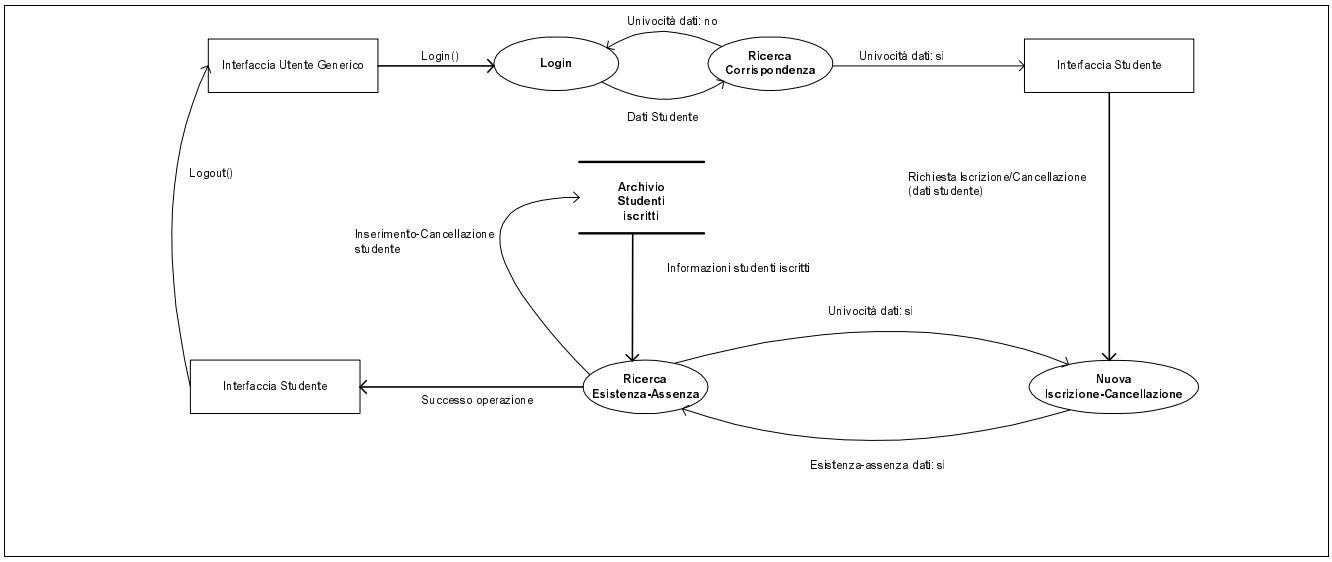


Figura III.21: DFD di uno studente

3.9 Package Diagram

Questo diagramma serve per avere una visione d'insieme di tutta l'applicazione.

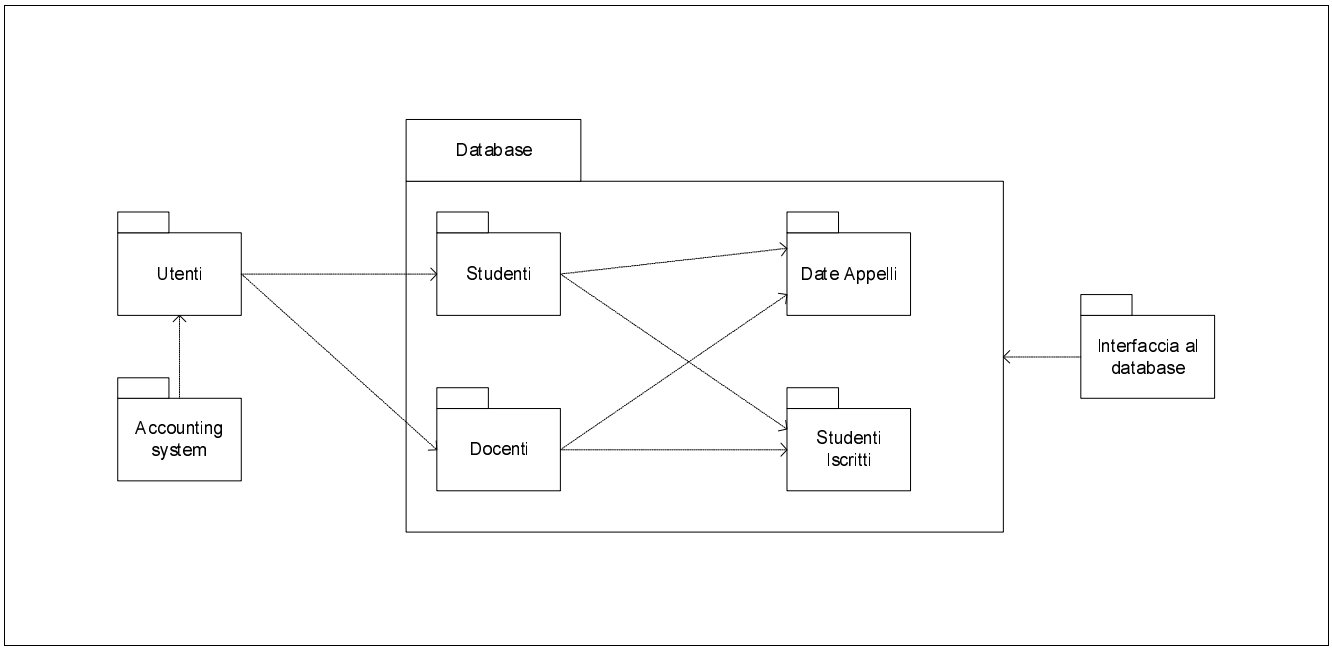


Figura III.22: Package Diagram

Capitolo 4

Applicazione realizzata

Nei capitoli precedenti abbiamo analizzato come, da un punto di vista teorico, si realizzi un'applicazione J2EE. In questo capitolo viene presentata la realizzazione pratica dell'applicazione per la gestione degli appelli d'esame.

Nello sviluppare questa applicazione si è reso necessario l'introduzione di nuove tabelle la cui struttura ricalca esattamente quella delle classi presentate nel class diagram. In questo modo è stata creata una precisa corrispondenza fra i record delle tabelle e le istanze delle classi usate all'interno del programma. Per sviluppare l'applicazione sono state utilizzati due tipi principali di strumenti: JSP (Java Server Pages) e EJB (Enterprise Java Bean). Per una migliore visione dell'applicazione procederemo attraverso di essa vedendo quali sono le diverse pagine che sono state realizzate ponendo particolare attenzione sugli EJB che sono stati utilizzati al loro interno. Per evitare di dilungarci troppo sul codice si è deciso di riportare soltanto la class bean, essendo la piú significativa . Al fine di comprendere meglio lo sviluppo dell'applicazione procediamo scindendola in tre parti in corrispondenza ai tre tipi di utente che possono accedere ad essa.

4.1 Utente generico

L'applicazione si apre con un pagina di ricerca dinamica in cui l'utente generico può ottenere informazioni riguardo agli appelli d'esame scegliendo fra le diverse opzioni messe a disposizione: cognome del docente, corso di studio, sessione o insegnamento. A questo punto entra in gioco un bean che prende in ingresso i dati scelti dallo studente e procede accedendo al database a ricavare tutti quegli appelli con le relative informazioni che rispondono ai requisiti richiesti.

Paginaricerca.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
                "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
  <title>Pagina ricerca</title>
```

```
  <!--<style type="text/css">
      </style>
  <meta http-equiv="Content-Type" content="text/html">
   <link rel="stylesheet" href="/StyleSheets/unimo3.css" type="text/css">-->
</head>

<%@ page language="java" import ="java.util.*" %>
<%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject, java.rmi.RemoteException,
                          esame.Dataappello, esame.DataappelloHome" %>


<%String corso=request.getParameter("corsi");
String cognome=request.getParameter("cognome");
String insegnamento=request.getParameter("insegnamento");
String sessione=request.getParameter("sessione");%>
<%Vector denominazione=new Vector();
Vector denominazionepubblica=new Vector();%>


<%try
     {

        Context jndiContext = new InitialContext();


        Object ref  = jndiContext.lookup("Dataappello");

        DataappelloHome home = (DataappelloHome)
                                        PortableRemoteObject.narrow (ref, DataappelloHome.class);


        Dataappello con = home.create();


        denominazione=
        con.getDataricercata(insegnamento,cognome,corso,sessione);

     }
     catch(Exception e)
     {
       System.out.println(e.toString());
     }%>

 <%if (denominazione.isEmpty()) {%><jsp:forward page="Erroredataricerca.html"/>

<%}%>
...
<td align="left" valign="top">
QUESTE SONO LE DATE ATTUALMENTE DISPONIBILI IN RETE:
<ul>
 <% for (int i=0;i<denominazione.size()-1;i=i+2){ %>
<li> <%=(String)denominazione.get(i+1)%>
 <% } %>
</ul>
<FORM METHOD="POST" ACTION=Paginagenerica.jsp>
<INPUT TYPE="submit" NAME="submit2" VALUE="Torna alla pagina precedente">
</FORM>
<FORM METHOD="POST" ACTION=Introduzione.html>
<INPUT TYPE="submit" NAME="submit2" VALUE="Login/Registrazione">
</FORM>
</tr>
</table>
...
</body>
</html>
```

DataappelloBean.java

```
package esame;
```

```
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.sql.*;
import java.io.*;
import java.beans.*;
import javax.servlet.http.*;
import javax.servlet.*;
import javax.sql.*;
import javax.naming.InitialContext;
import java.util.*;
import java.text.*;

public class DataappelloBean implements SessionBean
 {
public Vector getDataricercata(String insegnamento,String cognome,String corso,String sessione)

{

	Calendar  calendar = null;
	Vector dataappello1=new Vector();
	String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";

	Connection con;
	String createString;
	createString = "SELECT DISTINCT Dataappello,Idappello,TBL_INSEGNAMENTI.Denominazione AS ins,
TBL_DOCENTI.Cognome AS Cognome,TBL_CORSI.Denominazione AS corso1,Aula,Ora,Chiusuragiorni FROM
TBL_CORSI JOIN TBL_STUDENTI ON (TBL_CORSI.Id=TBL_STUDENTI.Idcorso)
JOIN TBL_CORSO_STUDIO_REGOLAMENTO ON
(TBL_CORSI.Id=TBL_CORSO_STUDIO_REGOLAMENTO.IdCorsoDiStudio) JOIN TBL_CORSI_AA ON
(TBL_CORSI_AA.IdCorsoStudioRegolamento=TBL_CORSO_STUDIO_REGOLAMENTO.Id) JOIN
TBL_INSEGNAMENTI_AA_PERIODI ON (TBL_INSEGNAMENTI_AA_PERIODI.IdCorsoAA=TBL_CORSI_AA.Id) JOIN
TBL_INSEGNAMENTI_AA ON (TBL_INSEGNAMENTI_AA.IdPeriodo=TBL_INSEGNAMENTI_AA_PERIODI.Id) JOIN
TBL_INSEGNAMENTI  ON (TBL_INSEGNAMENTI.Id=TBL_INSEGNAMENTI_AA.Idinsegnamento) JOIN
TBL_DATE_APPELLI ON (TBL_DATE_APPELLI.Idinsegnamento=TBL_INSEGNAMENTI.Id) JOIN
 TBL_AULE ON (TBL_DATE_APPELLI.Idaula=TBL_AULE.Idaula)  JOIN
TBL_DOCENTI_INSEGNAMENTI_AA ON
(TBL_INSEGNAMENTI_AA.Id=TBL_DOCENTI_INSEGNAMENTI_AA.IdInsegnamentoAA)
JOIN TBL_DOCENTI ON (TBL_DOCENTI_INSEGNAMENTI_AA.IdDocente=TBL_DOCENTI.Id)
WHERE TBL_DOCENTI.Cognome LIKE '" + cognome + "' OR TBL_INSEGNAMENTI.Denominazione
LIKE '" + insegnamento + "' OR TBL_DATE_APPELLI.Idsessione='" + sessione + "' OR
TBL_CORSI.Id='" + corso + "'";

	Statement stmt;

	try {
		InitialContext jndiContext = new InitialContext();


		DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");

		con = ds.getConnection();
		stmt = con.createStatement();

		ResultSet rs;

		int k=0;

		rs = stmt.executeQuery(createString);


		java.util.Date data1=null;

		String aulaappello="";

		String oraappello="";


 java.util.Date dataattuale=new java.util.Date();

		while (rs.next()) {
```

```
                    data1=rs.getDate("Dataappello");
                    k=rs.getInt("Chiusuragiorni");

                    String      insegnamento1=rs.getString("ins");

                    aulaappello=rs.getString("Aula");
                    oraappello=rs.getString("Ora");

                    calendar = Calendar.getInstance();

                    calendar.setTime (data1);

                    calendar.set (Calendar.DAY_OF_MONTH,

                    calendar.get(Calendar.DAY_OF_MONTH) - k);

                    java.util.Date dataprecedente= calendar.getTime();







       if ((dataprecedente.compareTo(dataattuale))>0) {

                                        SimpleDateFormat formatter= new SimpleDateFormat ("dd/MM/yy");


                                        String dateString = formatter.format(data1);

                                        String dateString1 = formatter.format(dataprecedente);


                                        String cognome1=rs.getString("Cognome");

                                        String corso1=rs.getString("corso1");

                                        Integer num=new Integer(rs.getInt("Idappello"));

                                        dataappello1.add(num);

dataappello1.add(insegnamento1+"-"+dateString +"-"+ aulaappello +"-Ora:"+ oraappello+"-ChiusuraLista:"+
dateString1+"-Prof."+cognome1+"-"+corso1);}

             }

      }  catch(Exception ex) {
         System.out.println("ERRORE in getConnection o in executeQuery:");
         System.err.println("SQLException: " + ex.getMessage());
      }

   return (dataappello1);}

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

Le informazioni che vengono messe a disposizione dell'utente vengono ricavate da una tabella che raccoglie le informazioni principali sugli appelli.

TBL_DATE_APPELLI

- ***Idappello:*** identifificatore dell'appello;
- ***Idinsegnamento:*** identificatore dell'insegnamento;
- ***Dataappello:*** data in cui si svolge l'appello;
- ***ChiusuraGiorni:*** numero dei giorni a partire dai quali viene ad essere chiusa la lista d'iscrizione ad un esame;
- ***Idsessione:*** identificatore della sessione di appartenenza dell'appello d'esame;
- ***Idaula:*** identificatore dell'aula in cui si svolge l'esame;
- ***Ora:*** ora in cui si svolge l'esame.

Nome del campo	Tipo di dato
Idappello	Integer
Idinsegnamento	Integer
Dataappello	Datetime
ChiusuraGiorni	Integer
Idsessione	Integer
Idaula	Integer
Ora	String

Dopo aver compiuto una ricerca di questo tipo l'utente generico può avere la necessità di compiere operazioni di più alto livello. Per poter ottenere informazioni di livello superiore è necessario che l'utente compia un'operazione di login. Durante questa operazione vengono richiesti username e password propri dell'utente. Nel caso in cui l'utente generico sia uno studente, quest'ultimo se sprovvisto di account, può accedere ad una pagina dedicata che gli permette, immettendo i propri dati anagrafici, di iscriversi alla lista che contiene gli studenti che hanno un account per poter accedere a questa applicazione. In questo caso entra in gioco un bean chiamato registrazione che si occupa di aggiungere a questa lista ogni studente che lo desideri. Nel caso in cui la registrazione sia già avvenuta non si procede alla registrazione e si mostra l'errore.

Paginaregistrazione.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
                "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
```

```
<head>
  <title>Pagina registrazione</title>
  <!--<style type="text/css">    </style>

  <meta http-equiv="Content-Type" content="text/html">
   <link rel="stylesheet" href="/StyleSheets/unimo3.css" type="text/css">-->

</head>
<%@ page language="java" import ="java.util.Vector" %>
<%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject,
java.rmi.RemoteException, esame.Corsi,esame.CorsiHome" %>
 <%Vector denominazione=new Vector();%>
<%try
     {
       Context jndiContext = new InitialContext();

       Object ref  = jndiContext.lookup("Corsi");

       CorsiHome home = (CorsiHome)
          PortableRemoteObject.narrow (ref, CorsiHome.class);

       Corsi con = home.create();

       denominazione= con.getCorsi();

     }
     catch(Exception e)
     {
       System.out.println(e.toString());
     }%>
...

<td align="left" valign="top">
<font size=5>
<FORM METHOD="POST" ACTION=Registrazione.jsp>
<font size=5>Cognome<br><input type=text name="cognome" size=20>
</font>
<br>
<font size=5>Nome<br><input type=text name="nome" size=20>
</font>
<br>
<font size=5>Matricola<br><input type=text name="numeromatricola" size=20>
</font>
<br>
<font size=5>Username<br><input type=text name="username" size=20>
</font>
<br>
<font size=5>Password<br><input type=password name="password" size=20>
</font>
<br>
<font size=5>E-Mail<br><input type=text name="email" size=20>
</font>
<br>
<select name="corsi">
 <% for (int i=0;i<denominazione.size()-1;i=i+2){ %>
 <option
VALUE="<%=(Integer)denominazione.get(i)%>"><%=(String)denominazione.get(i+1)%>
 <% } %>
</select>
<INPUT TYPE="submit" NAME="submit3" VALUE="Registrazione">
</FORM><br>
<FORM METHOD="POST" ACTION=Introduzione.html>
<INPUT TYPE="submit" NAME="submit3" VALUE="Esci">
</FORM>
</tr>
</table>
...
</body>
</html>
```

RegistrazioneBean.java

```
...

 public class RegistrazioneBean implements SessionBean
 {

     public int Registrastudente(String cognome,String nome,String username,String password,String nummat,String email,
                                          String corso)

       {

       String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";

       Connection con;
       String createString;
      createString = "INSERT INTO TBL_STUDENTI(Cognome,Nome,username,Passwd,Nummat,E_MAIL,Idcorso)
VALUES('" +cognome + "','" + nome + "','" + username + "','" + password + "','" + nummat + "',
                        '" + email + "','" + corso + "')";
       Statement stmt;
       int rs=0;

       try {

            InitialContext jndiContext = new InitialContext();


            DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");

            con = ds.getConnection();

            stmt = con.createStatement();

            rs = stmt.executeUpdate(createString);


            stmt.close();


            con.close();

           }  catch(Exception ex) {

              System.out.println("ERRORE in getConnection o in executeQuery:");

              System.err.println("SQLException: " + ex.getMessage());
       }
    return(rs);
    }
...
```

La tabella che contiene tutte le informazioni degli studenti che hanno regolare account e che verrà utilizzata al momento della verifica dello username e password immessi è cosí definita.

TBL_STUDENTI

- ***Idstudente:*** identificatore dello studente;
- ***Cognome:*** cognome dello studente;
- ***Nome:*** nome dello studente;
- ***NumeroMatricola:*** numero di matricola dello studente;
- ***Username:*** username dello studente;
- ***Password:*** password dello studente;
- ***Idcorso:*** corso di appartenenza dello studente;
- ***E-Mail:*** indirizzo E-Mail dello studente.

Nome del campo	Tipo di dato
Idstudente	Integer
Cognome	String
Nome	String
Nummat	String
Username	String
Password	String
Idcorso	Integer
E-Mail	String

4.2 Studente

Da qui in poi ciascun utente avrà la possibilità di compiere soltanto un certo numero di operazioni proprie del ruolo che ricopre. Cominciamo con il considerare il ruolo di studente. Nella prima pagina messa a disposizione dello studente viene fornita la lista degli insegnamenti che appartengono al suo corso di insegnamento e che sui quali può ottenere informazioni e la lista delle sessione di esame attualmente disponibili.

Dopo aver scelto l'insegnamento del quale vuole ottenere le informazioni si apre una pagina all'interno del quale vengono messe a disposizione dello studente tre diverse funzionalità. La prima è quella della possibilità di visualizzare il numero degli studenti iscritti ad un determinato esame.

Visualizza.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
                "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
<title>Visualizza numero studenti iscritti</title>
<!--<style type="text/css">
</style>
<meta http-equiv="Content-Type" content="text/html">
<link rel="stylesheet" href="/StyleSheets/unimo3.css" type="text/css">-->
```

```
</head>

<%@ page language="java"%>
<%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject,
java.rmi.RemoteException, esame.Conta1, esame.Conta1Home" %>

<%  String appello=request.getParameter("appello");
 int i=0;%>


 <% try
     {
       // Get a naming context
       Context jndiContext = new InitialContext();

       // Get a reference to a CD Bean
       Object ref  = jndiContext.lookup("Conta1");

       // Get a reference from this to the Bean's Home interface
       Conta1Home home = (Conta1Home)
          PortableRemoteObject.narrow (ref, Conta1Home.class);


       Conta1 con = home.create();


       i = con.getNumeroiscritti (appello);



     }
     catch(Exception e)
     {
       System.out.println(e.toString());
     }%>


...

<td align="left" valign="top">
<center><font size=5>Il numero degli iscritti e':</font>
 <%=i%></center>
<FORM METHOD="POST" ACTION=Paginastudente2.jsp>
<INPUT TYPE="submit" NAME="submit2" VALUE="Torna alla pagina precedente">
</FORM>
<FORM METHOD="POST" ACTION=Introduzione.html>
<INPUT TYPE="submit" NAME="submit1" VALUE="Esci">
</FORM>
</tr>
</table>
...

</body>
</html>
```

Conta1Bean.java

```
...

 public class Conta1Bean implements SessionBean
 {

      public int getNumeroiscritti(String data) {

       String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";

       Connection con;
       String createString;
```

```
    createString = "SELECT * FROM TBL_STUDENTIISCRITTI WHERE Idappello='" + data + "'";
    Statement stmt;
    int i=0;


    try {

       InitialContext jndiContext = new InitialContext();

       DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
       con = ds.getConnection();
       stmt = con.createStatement();
       ResultSet rs;
       rs = stmt.executeQuery(createString);
       while (rs.next()) { i++; }
       rs.close();
       stmt.close();

       con.close();

    } catch(Exception ex) {
       System.out.println("ERRORE in getConnection o in executeQuery:");
       System.err.println("SQLException: " + ex.getMessage());
    }

 return (i);
 }

...
```

La seconda da la possibilità allo studente di visualizzare i voti conseguiti in appelli precedentemente sostenuti.

Visualizzavoti.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
                "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
```

```
  <title>Visualizza voti</title>
  <!--<style type="text/css">
      </style>
  <meta http-equiv="Content-Type" content="text/html">
   <link rel="stylesheet" href="/StyleSheets/unimo3.css" type="text/css">-->
</head>
<%@ page language="java" import ="java.util.Vector" %>
<%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject,
java.rmi.RemoteException, esame.Elencoiscritti, esame.ElencoiscrittiHome" %>

<%Vector denominazione=new Vector();
String data=request.getParameter("appellopubblico");
String username=(String)session.getValue("usr");
String password=(String)session.getValue("psw");
String utente=(String)session.getAttribute("utente");%>

<%  if (utente.equals("studente")){   try
     {

        Context jndiContext = new InitialContext();


        Object ref  = jndiContext.lookup("Elencoiscritti");

        ElencoiscrittiHome home = (ElencoiscrittiHome)
           PortableRemoteObject.narrow (ref, ElencoiscrittiHome.class);


        Elencoiscritti con = home.create();


        denominazione= con.getElencosingolo(data,username,password);



     }
     catch(Exception e)
     {
       System.out.println(e.toString());
     }} else { %> <jsp:forward page="Errorechiusura.html"/> <%}%>
    <% if (denominazione.isEmpty()) { %> <jsp:forward page="Nonvoto.html"/>
<%}%>


...
<td align="left" valign="top">
<table BORDER COLS=3 WIDTH="100%" NOSAVE >
<tr>
<td>COGNOME</td>
<td>NOME</td>
<td>NUM.MATR.</td>

<td>ES.1</td>

<td>ES.2</td>

<td>ES.3</td>
<td>ES.4</td>
<td>ES.5</td>
<td>VOTO FINALE</td>
</center>
</tr>
<% for (int i=0;i<denominazione.size()-8;i=i+9){ %>
<tr>
<td><center><%=(String)denominazione.get(i) %></td>
<td><center><%=(String)denominazione.get(i+1) %></td>
<td><center><%=(String)denominazione.get(i+2) %></td>
<td><center><%=(String)denominazione.get(i+3) %>  </td>
<td><center><%=(String)denominazione.get(i+4) %></td>
<td><center><%=(String)denominazione.get(i+5) %></td>
<td><center><%=(String)denominazione.get(i+6) %></td>
<td><center><%=(String)denominazione.get(i+7) %></td>
```

```
<td><center><%=(String)denominazione.get(i+8) %></td>
</tr>
<% } %>

</table>
 <FORM METHOD="POST" ACTION=Paginastudente2.jsp>
<INPUT TYPE="submit" NAME="submit" VALUE="Ritorna">
</FORM>

</tr>
</table>
<hr>
...
</body>
</html>
```

ElencoiscrittiBean.java

```
...

 public class ElencoiscrittiBean implements SessionBean
 {

        public Vector getElencosingolo (String data,String username,String password)

      {
       Vector denominazione2=new Vector();
       String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";

       Connection con;
       String createString;
       createString = "SELECT * FROM TBL_STUDENTI JOIN TBL_STUDENTIISCRITTI ON
                       (TBL_STUDENTI.Idstudente=TBL_STUDENTIISCRITTI.Idstudente)
                       WHERE  username='" +          username + "' AND Passwd='" + password + "' AND Idappello='" + data + "'";
       Statement stmt;

       try {

              InitialContext jndiContext = new InitialContext();

          DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");

              con = ds.getConnection();

              stmt = con.createStatement();

              ResultSet rs;

              rs = stmt.executeQuery(createString);

             while (rs.next()) {

                              denominazione2.add(rs.getString("Cognome"));


                              denominazione2.add(rs.getString("Nome"));


                              denominazione2.add(rs.getString("Nummat"));

                              denominazione2.add(rs.getString("ES1"));


                              denominazione2.add(rs.getString("ES2"));

                              denominazione2.add(rs.getString("ES3"));

                              denominazione2.add(rs.getString("ES4"));

                              denominazione2.add(rs.getString("ES5"));
```

```
                        denominazione2.add(rs.getString("Votofinale"));

                    }

                rs.close();
                stmt.close();
                con.close();

            } catch(Exception ex) {
                System.out.println("ERRORE in getConnection o in executeQuery:");
                System.err.println("SQLException: " + ex.getMessage());
            }
            return (denominazione2);
        }
...
```

La terza ed ultima funzionalità messa a disposizione dello studente è quella che permette di iscriversi o cancellarsi da una lista d'esame.

Iscrizcanc.jsp

```
<HTML>
<HEAD>
<TITLE>
Iscrizcancellazione
</TITLE>
</HEAD>
<BODY>

<%@ page language="java" import ="java.util.*" %>

 <%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject, java.rmi.RemoteException,

                                                esame.Iscrizcanc,esame.IscrizcancHome,esame.Controllo,esame.ControlloHome" %>

<%String username=(String)session.getValue("usr");
String password=(String)session.getValue("psw");%>


<%String appello=request.getParameter("appello");
  String scelta=request.getParameter("isccanc");
  session.setAttribute("scelta",scelta);
  int stato=0;
  int Ok=0;
  int Idinsegnamento=0;%>

<%String utente=(String)session.getValue("utente");

     try
      {



 Context jndiContext = new InitialContext();

       Object ref  = jndiContext.lookup("Controllo");



 ControlloHome home = (ControlloHome)  PortableRemoteObject.narrow (ref, ControlloHome.class);

       Controllo con = home.create();
```

```
      stato= con.getControllo(username,password);


      }
      catch(Exception e)
      {
        System.out.println(e.toString());
      }
      if (scelta.equals("Iscriviti")) {

       try
      {

       Context jndiContext = new InitialContext();

       Object ref  = jndiContext.lookup("Iscrizcanc");


       IscrizcancHome home = (IscrizcancHome)
       PortableRemoteObject.narrow (ref, IscrizcancHome.class);


       Iscrizcanc con = home.create();
Ok=    con.Iscrivistudente(stato,appello);


      }
      catch(Exception e)
      {
        System.out.println(e.toString());
      }}else if (scelta.equals("Ritirati"))    {


      try
      {

        Context jndiContext = new InitialContext();

        Object ref  = jndiContext.lookup("Iscrizcanc");

                                IscrizcancHome home = (IscrizcancHome)    PortableRemoteObject.narrow (ref, IscrizcancHome.class);

        Iscrizcanc con = home.create();
        Ok=    con.Ritirostudente(stato,appello);
      }
      catch(Exception e)
      {
        System.out.println(e.toString());
      }}%>
<% String stato1="";
     Integer num=new Integer(stato);

     stato1=num.toString();
     session.setAttribute("stato",stato1);%>

<% if (Ok==0){ %> <jsp:forward page="ErroreSQL.html"/> <%} else {%> <jsp:forward page="OK.jsp"/> <%}%>
</BODY>

</HTML>
```

IscrizcancBean.java

```
...
 public class IscrizcancBean implements SessionBean
 {

     public int Iscrivistudente(int studente,String id) {

        int idappello=Integer.parseInt(id);

       String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";
       Connection con;
```

```
      String createString;
      createString = "INSERT INTO TBL_STUDENTIISCRITTI(Idstudente,Idappello,ES1,ES2,ES3,ES4,ES5,
Votofinale,Pubblica) VALUES('" + studente + "','" + idappello + "',0,0,0,0,0,0,null)";
      Statement stmt;
      int rs=0;
      try {

         InitialContext jndiContext = new InitialContext();
         DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
         con = ds.getConnection();
         stmt = con.createStatement();
         System.out.println("statement beccato");
         rs = stmt.executeUpdate(createString);

         stmt.close();

         con.close();
      }  catch(Exception ex) {
         System.out.println("ERRORE in getConnection o in executeQuery:");
         System.err.println("SQLException: " + ex.getMessage());
      }
   return(rs);
   }

      public int Ritirostudente(int id,String idappello) {

      String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";

      Connection con;
      String createString;
      createString = "DELETE FROM TBL_STUDENTIISCRITTI WHERE Idstudente='" + id + "' AND
                      Idappello='" + idappello + "'";
      Statement stmt;

      int rs=0;

      try {

      InitialContext jndiContext = new InitialContext();

         DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
         con = ds.getConnection();
         stmt = con.createStatement();
         rs=stmt.executeUpdate(createString);

         stmt.close();

         con.close();


      }  catch(Exception ex) {
         System.out.println("ERRORE in getConnection o in executeQuery:");
         System.err.println("SQLException: " + ex.getMessage());
      }

        return(rs);

        }
...
```

Di seguito vediamo la tabella che contiene tutte quante le informazioni riguardo agli studenti iscritti ad un determinato appello.

TBL_STUDENTIISCRITTI

- ***Idstudente:*** identificatore dello studente;
- ***Idappello:*** identificatore dell'appello al quale lo studente è iscritto;
- ***ES1:*** voto conseguito nell'esercizio numero 1 da parte dello studente;
- ***ES2:*** voto conseguito nell'esercizio numero 2 da parte dello studente;
- ***ES3:*** voto conseguito nell'esercizio numero 3 da parte dello studente;
- ***ES4:*** voto conseguito nell'esercizio numero 4 da parte dello studente;
- ***ES5:*** voto conseguito nell'esercizio numero 5 da parte dello studente;
- ***Voto finale:*** voto complessivo degli esercizi sostenuti dallo studente;
- ***Pubblica:*** possibilità o meno di mettere a disposizione da parte di un docente a disposizione della consultazione da parte degli studenti.

Nome del campo	Tipo di dato
Idstudente	Integer
Idappello	Integer
ES1	Integer
ES2	Integer
ES3	Integer
ES4	Integer
ES5	Integer
Voto finale	Integer
Publica	String

4.3 Docente

Veniamo ora a prendere in considerazione le funzionalità offerte al docente. Come nel caso dello studente al docente viene presentata una schermata iniziale in cui vengono messi a disposizione soltanto gli insegnamenti tenuti da quel docente e le diverse sessioni messe a disposizione. Una volta scelto l'insegnamento si apre un pagina in cui vengono messe a disposizione le funzionalità offerte al docente dall'applicazione. La prima funzionalità offerta è quella per la quale è possibile visualizzare il numero

degli studenti iscritti ad esame. La seconda è quella che permette di mutare il giorno di chiusura della lista di esame.

Modificachiusura.jsp

```
<HTML>
<HEAD>
<TITLE>
Modifica chiusura
</TITLE>
</HEAD>
<BODY>
```

```
<%@ page language="java"%>

 <%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject, java.rmi.RemoteException,
                                        esame.Chiusuraappello, esame.ChiusuraappelloHome" %>

<%int chiuso=0;%>


<%String appello=request.getParameter("appello");%>
<%String chiusura=request.getParameter("Chiusuraappello");%>

<% String utente=(String)session.getAttribute("utente");

    try
      {

         Context jndiContext = new InitialContext();

         Object ref  = jndiContext.lookup("Chiusuraappello");



        ChiusuraappelloHome home = (ChiusuraappelloHome)  PortableRemoteObject.narrow
(ref, ChiusuraappelloHome.class);

        Chiusuraappello con = home.create();


        chiuso=   con.getModifica(appello,chiusura);


      }
      catch(Exception e)
      {
        System.out.println(e.toString());
      }%>
 <% if (chiuso==0) { %> <jsp:forward page="ErroreSQLchiusura.html"/> <%} else {%>
                                                                    <jsp:forward page="Paginadocente2.jsp"/> <%}%>

</BODY>
</HTML>
```

ChiusuraappelloBean.java

```
...
 public class ChiusuraappelloBean implements SessionBean
 {

     public int getModifica(String data,String chiusura) {
 int n=Integer.parseInt(chiusura);
  int datai=Integer.parseInt(data);

        String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";
        Connection con;
        String createString;
        createString = "UPDATE TBL_DATE_APPELLI  SET Chiusuragiorni='" + n + "'
WHERE Idappello='" + datai + "'";
        Statement stmt;
int rs=0;



        try {
```

```
          InitialContext jndiContext = new InitialContext();

        DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
        con = ds.getConnection();
        stmt = con.createStatement();
        System.out.println("statement beccato");
        rs = stmt.executeUpdate(createString);
                    System.out.println("chiuso con "+rs+ " righe");

        stmt.close();

        con.close();
      } catch(Exception ex) {
        System.out.println("ERRORE in getConnection o in executeQuery:");
        System.err.println("SQLException: " + ex.getMessage());
      }
            System.out.println(createString);

    return (rs);
    }

...
```

La terza è quella che permette di osservare quali sono gli studenti iscritti ad una determinata lista d'esame quando ancora essa è aperta. Inoltre c'è la possibilità di visualizzare la lista d'iscrizione chiusa e di eliminare dalla lista quegli studenti che pur essendosi iscritti non hanno sostenuto la prova. C'è la possibilità di immettere i voti che hanno conseguito gli studenti durante la prova. Infine c'è la possibilità di mettere a disposizione la lista con i voti agli studenti che hanno sostenuto l'esame.

Situazione.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
                    "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
  <title>Visualizza voti studente</title>
  <!--<style type="text/css">
        </style>
  <meta http-equiv="Content-Type" content="text/html">
   <link rel="stylesheet" href="/StyleSheets/unimo3.css" type="text/css">-->
</head>
<%@ page language="java" import ="java.util.Vector" %>
 <%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject,
java.rmi.RemoteException, esame.Elencoiscritti, esame.ElencoiscrittiHome" %>
```

```
<%Vector denominazione=new Vector();
String data=request.getParameter("appello");
session.putValue("data",data);
String utente=(String)session.getAttribute("utente");%>


<%try
     {

        Context jndiContext = new InitialContext();

        Object ref  = jndiContext.lookup("Elencoiscritti");

        ElencoiscrittiHome home = (ElencoiscrittiHome)
          PortableRemoteObject.narrow (ref, ElencoiscrittiHome.class);


        Elencoiscritti con = home.create();


        denominazione= con.getElenco(data);



     }
     catch(Exception e)
     {
       System.out.println(e.toString());
     }%>







<%if (denominazione.isEmpty()) { %> <jsp:forward page="Erroreassenza.html"/>
<%}%>
...


<td align="left" valign="top">
<table BORDER COLS=3 WIDTH="100%" NOSAVE >
<tr>
<td>ID</td>
<td>COGNOME</td>
<td>NOME</td>
<td>NUMERO MATRICOLA</td>
```

```
</tr>
<% for (int i=0;i<denominazione.size()-3;i=i+4){ %>
<tr>
 <td><%=(Integer)denominazione.get(i) %></td>
  <td><%=(String)denominazione.get(i+1) %></td>
   <td><%=(String)denominazione.get(i+2) %></td>
   <td><%=(String)denominazione.get(i+3) %>  </td>


</tr>
<% } %>


</table>

<FORM METHOD="POST" ACTION=Dateiscritti.jsp>
<INPUT TYPE="submit" NAME="submit2" VALUE="Ritorna alla pagina precedente">
</FORM>
<FORM METHOD="POST" ACTION=Introduzione.html>
<INPUT TYPE="submit" NAME="submit1" VALUE="Esci">
</FORM>

</tr>
</table>
<hr>
...
</body>
</html>
```

Situazionemodificabile.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
                "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
  <title>Visualizza studenti</title>
  <!--<style type="text/css">
       </style>
  <meta http-equiv="Content-Type" content="text/html">
   <link rel="stylesheet" href="/StyleSheets/unimo3.css" type="text/css">-->
</head>
<%@ page language="java" import ="java.util.Vector" %>
 <%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject,
java.rmi.RemoteException, esame.Elencoiscritti, esame.ElencoiscrittiHome" %>

<%Vector denominazione=new Vector();
String data=request.getParameter("appello");
session.putValue("data",data);
String utente=(String)session.getAttribute("utente");%>
```

```
<%try
     {
       // Get a naming context
       Context jndiContext = new InitialContext();

       // Get a reference to a CD Bean
       Object ref  = jndiContext.lookup("Elencoiscritti");

       // Get a reference from this to the Bean's Home interface
       ElencoiscrittiHome home = (ElencoiscrittiHome)
          PortableRemoteObject.narrow (ref, ElencoiscrittiHome.class);


       Elencoiscritti con = home.create();


       denominazione= con.getElenco(data);



     }
     catch(Exception e)
     {
       System.out.println(e.toString());
     }%>

<%if (denominazione.isEmpty()) { %> <jsp:forward page="Erroreassenza.html"/>
<%}%>

...
<td align="left" valign="top">
<table BORDER COLS=3 WIDTH="100%" NOSAVE >
<tr>
<td>ID</td>
<td>COGNOME</td>
<td>NOME</td>
<td>NUMERO MATRICOLA</td>
</tr>
<% for (int i=0;i<denominazione.size()-3;i=i+4){ %>
<tr>


<td><%=(Integer)denominazione.get(i) %></td>
  <td><%=(String)denominazione.get(i+1) %></td>
   <td><%=(String)denominazione.get(i+2) %></td>
   <td><%=(String)denominazione.get(i+3) %>  </td>


</tr>
<% } %>

</table>
<FORM METHOD="POST" ACTION=Paginaconferma.jsp>
<font size=5>IDENTIFICATORE STUDENTE<input type=text name="Id" size=10>
</font>
<INPUT TYPE="submit" NAME="submit2" VALUE="Cancella studente dalla lista">
</FORM><br><br>
<FORM METHOD="POST" ACTION=Voti.jsp>
<INPUT TYPE="submit" NAME="submit2" VALUE="Procedi">
</FORM>

<FORM METHOD="POST" ACTION=Dateiscritti.jsp>
<INPUT TYPE="submit" NAME="submit2" VALUE="Ritorna alla pagina precedente">
</FORM>
<FORM METHOD="POST" ACTION=Introduzione.html>
<INPUT TYPE="submit" NAME="submit1" VALUE="Esci">
</FORM>

</tr>
</table>
...
```

```
</body>
</html>
```

ElencoiscrittiBean.java

```
public class ElencoiscrittiBean implements SessionBean
 {

  public Vector getElenco (String data) {

       Vector denominazione2=new Vector();
       String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";

       Connection con;
       String createString;


createString = "SELECT * FROM TBL_STUDENTI JOIN TBL_STUDENTIISCRITTI ON
                    (TBL_STUDENTI.Idstudente=TBL_STUDENTIISCRITTI.Idstudente) WHERE  Idappello='" + data + "'";
       Statement stmt;

       try {

          InitialContext jndiContext = new InitialContext();

          DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
          con = ds.getConnection();
          stmt = con.createStatement();
          ResultSet rs;
          rs = stmt.executeQuery(createString);

          while (rs.next()) {
          Integer num=new Integer(rs.getInt("Idstudente"));

          denominazione2.add(num);

          denominazione2.add(rs.getString("Cognome"));

          denominazione2.add(rs.getString("Nome"));




denominazione2.add(rs.getString("Nummat"));

              }


          rs.close();
          stmt.close();


          con.close();

       }  catch(Exception ex) {
          System.out.println("ERRORE in getConnection o in executeQuery:");
          System.err.println("SQLException: " + ex.getMessage());
       }

    return (denominazione2);
    }

     public Vector getElencocompleto (String data) {

       Vector denominazione2=new Vector();
       String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433"
       Connection con;
       String createString;
       createString = "SELECT * FROM TBL_STUDENTI JOIN TBL_STUDENTIISCRITTI ON
                    (TBL_STUDENTI.Idstudente=TBL_STUDENTIISCRITTI.Idstudente) WHERE  Idappello='" + data + "'";
       Statement stmt;
```

```
      try {

         InitialContext jndiContext = new InitialContext();

         DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
         con = ds.getConnection();
         stmt = con.createStatement();
         ResultSet rs;
         rs = stmt.executeQuery(createString);
         while (rs.next()) {

         Integer num=new Integer(rs.getInt("Idstudente"));


        denominazione2.add(num);

        denominazione2.add(rs.getString("Cognome"));

        denominazione2.add(rs.getString("Nome"));

        denominazione2.add(rs.getString("Nummat"));
        denominazione2.add(rs.getString("ES1"));

        denominazione2.add(rs.getString("ES2"));
        denominazione2.add(rs.getString("ES3"));
        denominazione2.add(rs.getString("ES4"));
        denominazione2.add(rs.getString("ES5"));
        denominazione2.add(rs.getString("Votofinale"));

            }

         rs.close();
         stmt.close();

         con.close();

      }  catch(Exception ex) {
         System.out.println("ERRORE in getConnection o in executeQuery:");
         System.err.println("SQLException: " + ex.getMessage());
      }
      return (denominazione2);
   }
```

Cancellastudentelista.jsp

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>Cancella studente dalla lista iscritti</TITLE>
</HEAD>
<!--<BODY BACKGROUND="/esami/bkg/pink_fabric.gif">-->

<CENTER><H2>Facoltà di Ingegneria</H2><H2>Iscrizione agli esami</H2></CENTER>
<HR>
<%@ page language="java" import ="java.util.Vector" %>
<%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject, java.rmi.RemoteException,
                                esame.Cancellaiscritti, esame.CancellaiscrittiHome" %>
<% int fatto=0;
String data=(String)session.getAttribute("data");

String Idstudente=(String)session.getAttribute("Idstudente");

 try
     {

        Context jndiContext = new InitialContext();
```

```
        Object ref  = jndiContext.lookup("Cancellaiscritti");



        CancellaiscrittiHome home = (CancellaiscrittiHome) PortableRemoteObject.narrow
(ref, CancellaiscrittiHome.class);

        Cancellaiscritti con = home.create();

        fatto=con.getCancellaiscritti (Idstudente,data);


      }
      catch(Exception e)
      {
        System.out.println(e.toString());}%>

<jsp:forward page="Situazione2modificabile.jsp"/>

</BODY>
</HTML>
```

CancellaiscrittiBean.java

```
...
 public class CancellaiscrittiBean implements SessionBean
 {

    public int getCancellaiscritti(String Id,String data) {

       String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";
       int rs=1;
       Connection con;
       String createString;
       createString = "DELETE FROM TBL_STUDENTIISCRITTI WHERE Idstudente='" + Id + "' AND Idappello='" + data + "'";
       Statement stmt;


       try {

          InitialContext jndiContext = new InitialContext();

          DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
          con = ds.getConnection();
          stmt = con.createStatement();
          System.out.println("statement beccato");
          rs = stmt.executeUpdate(createString);
          System.out.println(createString);


stmt.close();

          con.close();


       } catch(Exception ex) {
          System.out.println("ERRORE in getConnection o in executeQuery:");
          System.err.println("SQLException: " + ex.getMessage());
       }

   return(rs);
   }

...
```

Modificavoti.jsp

```
<HTML>
<HEAD>
<TITLE>
Modifica voti</TITLE>
```

```
</HEAD>
<BODY>

<%@ page language="java"%>

 <%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject, java.rmi.RemoteException,
                                              esame.Modificavoti,esame.ModificavotiHome" %>

<%
  String id=request.getParameter("id");
  String es1=request.getParameter("es1");
  String es2=request.getParameter("es2");
  String es3=request.getParameter("es3");
  String es4=request.getParameter("es4");
  String es5=request.getParameter("es5");
  String vt=request.getParameter("vt");
  String data=(String)session.getAttribute("data");
  int rs=0;
%>


<%try
     {

        Context jndiContext = new InitialContext();

        Object ref  = jndiContext.lookup("Modificavoti");

       ModificavotiHome home = (ModificavotiHome) PortableRemoteObject.narrow (ref, ModificavotiHome.class);

        Modificavoti con = home.create();


        rs=  con.Modifica(id,data,es1,es2,es3,es4,es5,vt);


     }
     catch(Exception e)
     {
       System.out.println(e.toString());
     }%>

<% if (rs==0) {%> <jsp:forward page="ErroreSQLid.html"/> <%} else {%>
                                                                    <jsp:forward page="Voti.jsp"/> <%}%>

</BODY>
</HTML>
```

ModificavotiBean.java

```
...

 public class ModificavotiBean implements SessionBean
 {

    public int Modifica(String id,String data,String es1,String es2,String es3,String es4,String es5,String vt)
{
       int r=0;
      String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";

      Connection con;
      String createString;
      createString = "UPDATE  TBL_STUDENTIISCRITTI SET ES1='" + es1 + "',ES2='" + es2 + "',ES3='" + es3 + "',
ES4='" + es4 + "',ES5='" + es5 + "',Votofinale='" + vt + "'
                       WHERE Idappello='" + data + "' AND Idstudente='" + id + "'";


      Statement stmt;
```

```
      try {

         InitialContext jndiContext = new InitialContext();
         DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
         con = ds.getConnection();
         stmt = con.createStatement();
         r = stmt.executeUpdate(createString);
         stmt.close();
         con.close();

      } catch(Exception ex) {
         System.out.println("ERRORE in getConnection o in executeQuery:");
         System.err.println("SQLException: " + ex.getMessage());
      }

   return(r);
   }
...
```

Pubblica.jsp

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>Pubblica o cancella date</TITLE>
</HEAD>
<!--<BODY BACKGROUND="/esami/bkg/pink_fabric.gif">-->

<CENTER><H2>Facoltà di Ingegneria</H2><H2>Iscrizione agli esami</H2></CENTER>
<HR>

<%@ page language="java" import ="java.util.Vector" %>
 <%@ page import="javax.ejb.*, javax.naming.*, javax.rmi.PortableRemoteObject,
java.rmi.RemoteException, esame.Pubblica, esame.PubblicaHome" %>

<%
int rs=0;
String data=(String)session.getAttribute("data");



String scelta=request.getParameter("scelta");%>

<%if (scelta.equals("Pubblica data sul Web")){   try
      {

        Context jndiContext = new InitialContext();

        Object ref  = jndiContext.lookup("Pubblica");




 PubblicaHome home = (PubblicaHome)
           PortableRemoteObject.narrow (ref, PubblicaHome.class);
        Pubblica con = home.create();


        rs= con.Pubblicadata(data);



      }
      catch(Exception e)
      {
        System.out.println(e.toString());
      }}
     if (scelta.equals("Cancella data dal Web"))  {   try
      {

        Context jndiContext = new InitialContext();
```

```
        Object ref  = jndiContext.lookup("Pubblica");


        PubblicaHome home = (PubblicaHome)  PortableRemoteObject.narrow (ref, PubblicaHome.class);


        Pubblica con = home.create();

        rs= con.Cancelladata(data);



      }
      catch(Exception e)
      {
        System.out.println(e.toString());
      }}%>
<jsp:forward page="Dateiscritti.jsp"/>


</BODY>
</HTML>
```

PubblicaBean.java

```
public class PubblicaBean implements SessionBean
 {

    public int Pubblicadata(String data) {

       String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";
       int r=0;
       Connection con;
       String createString;
       createString = "UPDATE  TBL_STUDENTIISCRITTI SET Pubblica='Ok' WHERE Idappello='" + data + "'";
       Statement stmt;



try {

          InitialContext jndiContext = new InitialContext();

          DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
          con = ds.getConnection();
          stmt = con.createStatement();
          System.out.println("statement beccato");


          r = stmt.executeUpdate(createString);


          System.out.println(createString);
          stmt.close();
               con.close();

      }  catch(Exception ex) {
          System.out.println("ERRORE in getConnection o in executeQuery:");
          System.err.println("SQLException: " + ex.getMessage());
      }

   return(r);
   }
      public int Cancelladata (String data) {

      String url = "jdbc:microsoft:sqlserver://noumeno.ing.unimo.it:1433";
      int rs=0;
      Connection con;
      String createString;
```

```
createString = "UPDATE TBL_STUDENTIISCRITTI SET Pubblica='' WHERE Idappello='" + data + "'";
     Statement stmt;


     try {

         InitialContext jndiContext = new InitialContext();

        DataSource ds=(DataSource)jndiContext.lookup("java:/SQLServerPool");
        con = ds.getConnection();
        stmt = con.createStatement();

        rs=  stmt.executeUpdate(createString);


        stmt.close();

        con.close();



     }  catch(Exception ex) {
        System.out.println("ERRORE in getConnection o in executeQuery:");
        System.err.println("SQLException: " + ex.getMessage());
     }
        return(rs);

       }
```

Appendice

Manuale utente

Ora ci proponiamo di creare un manuale utente facendo riferimento ad alcune delle schermate che si possono presentare ad un utente percorrendo l'applicazione per la gestione degli appelli d'esame. Come è già accaduto nel corso dei capitoli precedenti analizzeremo l'applicazione considerando di volta in volta le funzionalità messe a disposizione dei diversi utenti. Cominciamo con il considerare l'utente generico che come abbiamo avuto già modo di dire è l'utente che ancora non ha compiuto alcuna

operazione di login e quindi la cui identità è sconosciuta al sistema. Nel momento in cui l'utente accede all'applicazione gli viene mostrata una pagina di ricerca dinamica in cui egli può scegliere se impostare la ricerca sulla base del nome dell'insegnamento, sul cognome del docente che tiene l'insegnamento, su una sessione o sul corso di studio.

Figura A.1: Pagina di ricerca dinamica

Dopo aver scelto di procedere vengono mostrate all'utente le informazioni riguardanti gli appelli che rispondono ai parametri di ricerca. Naturalmente è possibile che non venga visualizzata alcuna informazione. Come si ha modo di vedere dalla immagine seguente le informazioni che eventualmente vengono mostrate all'utente contengono i seguenti dati:

- Nome dell'insegnamento;

- Data in cui si svolge la prova d'esame;
- Aula in cui si svolge l'esame;
- Ora in cui si svolge l'esame;
- Data di chiusura della lista per l'iscrizione all'esame;
- Cognome del professore che tiene il corso;
- Corso di studio cui appartiene questo insegnamento.

Figura A.2: Pagina di visualizzazione dei dati richiesti

Dopo aver ottenuto queste informazioni all'utente può interessare compiere operazioni di più alto livello. Si rende necessario che l'utente compia un operazione di login. Si può presentare la possibilità che l'utente (in particolare lo studente) non possieda l'account per poter accedere all'applicazione. A questo punto l'utente può richiedere di registrarsi accedendo ad una pagina che permette, immettendo alcuni dati specifici, di farlo. Come si ha modo di osservare dall'immagine viene richiesto di immettere il cognome, nome, numero di matricola, username e password desiderati,

indirizzo E-Mail ed infine il corso di studi di appartenenza. Il sistema sulla base dei dati immessi procederà alla registrazione del nuovo utente. Nel caso la registrazione sia già avvenuta in passato il sistema mostrerà un messaggio di errore in cui si sottolinea la non correttezza dell'operazione che si richiede.

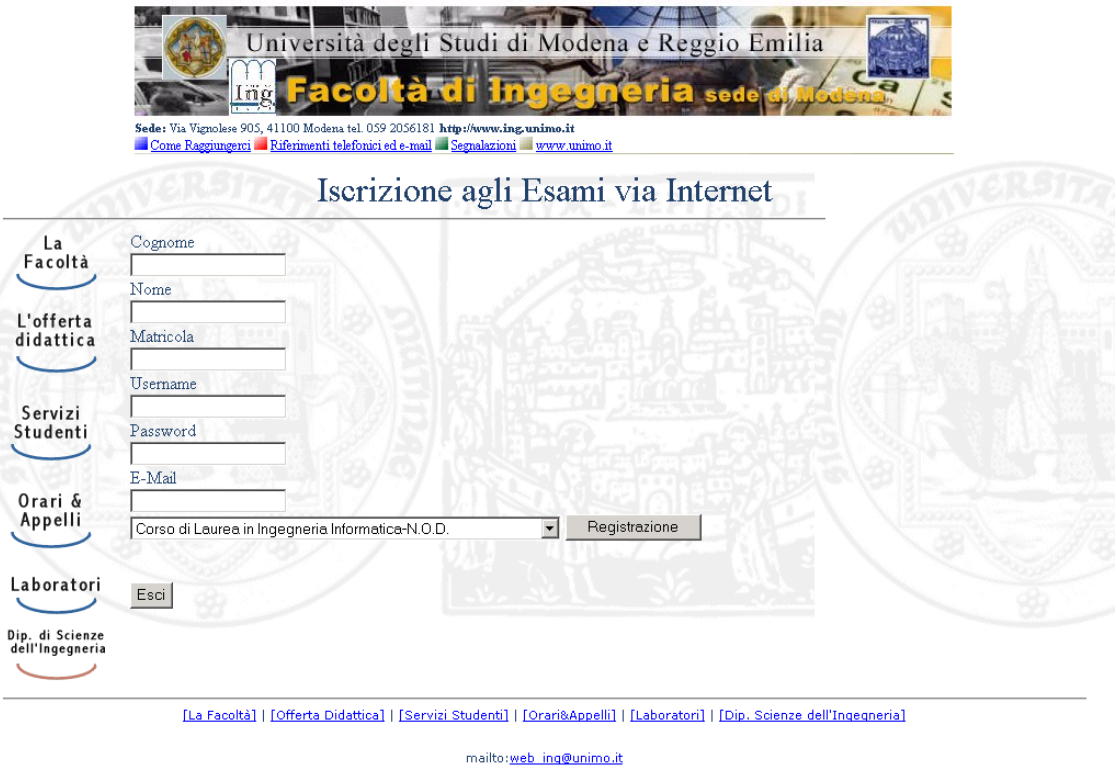


Figura A.3: Pagina di registrazione

Dopo essersi eventualmente registrati l'utente può decidere di compiere l'operazione di login che consiste nel immettere il proprio username e password. Il sistema in seguito all'immissione dei dati verifica l'esattezza dei dati immessi ed in caso affermativo assegnerà all'utente un ruolo: studente o docente. D'ora in poi sulla base del ruolo assegnato ciascun tipo di utente potrà compiere solo un numero ristretto di operazioni.

Figura A.4: Pagina di login

Dopo aver consentito l'accesso il sistema mostra all'utente gli insegnamenti sui quali può operare e le sessioni che sono attualmente disponibili. Nel caso in cui l'utente che accede sia uno studente gli vengono mostrati gli insegnamenti del suo corso di studio di appartenenza, nel caso sia un docente gli vengono mostrati gli insegnamenti che tiene.

Figura A.5: Pagina di scelta insegnamento e sessione

Proseguiamo la trattazione prendendo dapprima in considerazione le funzionalità messe a disposizione dello studente e poi quelle messe a disposizione del docente. Le operazioni consentite allo studente sono di tre tipi:

- Visualizzazione del numero degli studenti iscritti ad un determinato appello;
- Iscrizione/Cancellazione ad/da una lista d'esame;
- Visualizzazione dei voti conseguiti in esami già sostenuti.

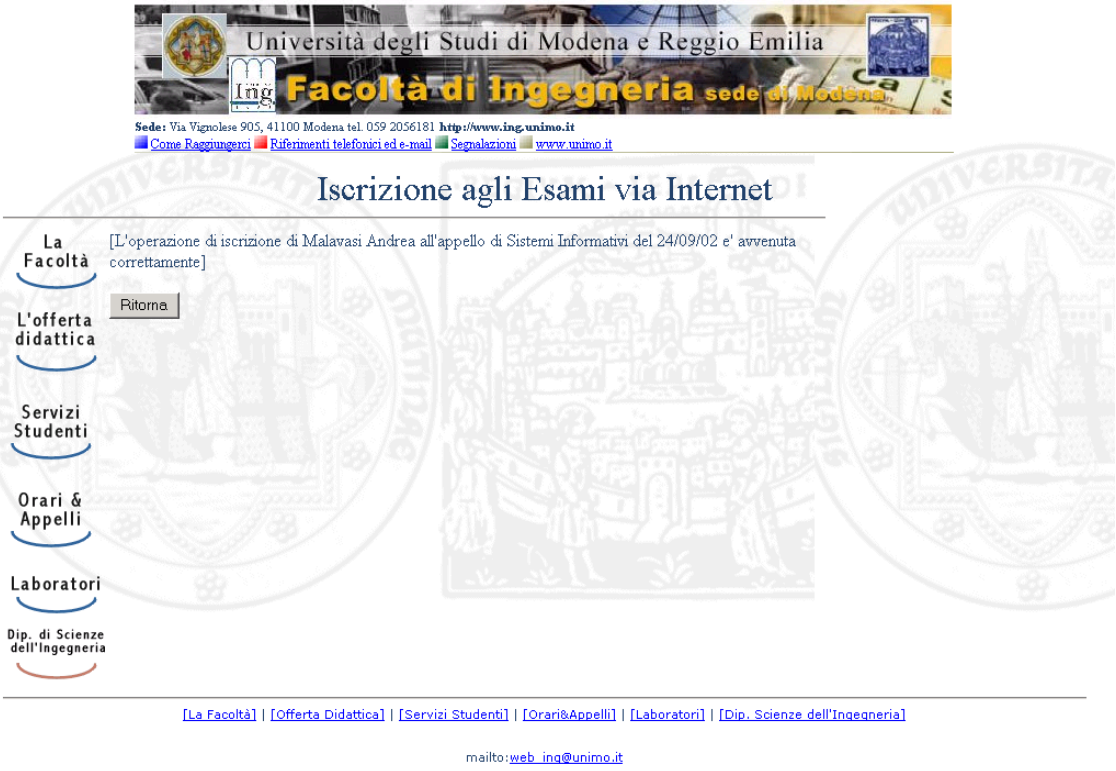
Figura A.6: Pagina di scelta-studente dell'operazione da eseguire

Una volta scelto l'appello di cui si vuole visualizzare il numero degli iscritti il sistema procede mostrando il numero degli studenti iscritti a quell'esame.

Figura A.7: Pagina di visualizzazione del numero degli studenti iscritti ad un esame

Nel momento in cui uno studente decide di volersi iscrivere/cancellare da una lista d'esame lo può fare semplicemente scegliendo l'appello che gli interessa e scegliendo fra le opzioni "Iscriviti" o "Ritirati". Quello che è interessante notare è che la lista d'esame ha un termine di scadenza che viene

mostrato dalla dicitura “Chiusura Lista”. Nel momento in cui viene superato questo limite la lista risulta non essere più a disposizione dello studente ma soltanto del docente. Nel momento in cui uno studente compie un’operazione di iscrizione/cancellazione il sistema si preoccupa di verificare che l’operazione compiuta non sia già stata fatta in passato. Nel caso ciò non avvenga il sistema mostra un messaggio in cui si sottolinea che lo studente si è regolarmente iscritto o cancellato.



A.8: Pagina di conferma dell’iscrizione di uno studente ad un esame

Dopo che un docente ha messo in rete la lista degli studenti che hanno sostenuto un esame con allegati i voti conseguiti da ciascuno di essi lo studente ha la possibilità, scegliendo l’appello di cui desidera ottenere l’informazione, di visualizzare i propri risultati.

Figura A.9: Pagina di visualizzazione dei voti conseguiti da uno studente

Prendiamo ora in considerazione le funzionalità messe a disposizione del docente. Dall'osservazione dell'immagine se ne possono soltanto osservare alcune. Per avere una visione complessiva bisogna procedere attraverso l'applicazione. Prima di procedere vediamone una visione d'insieme. Un docente può:

- Visualizzare il numero degli studenti iscritti ad un esame;

- Modificare il giorno di chiusura di una lista;
- Visualizzare gli studenti iscritti ad una lista d'esame quando ancora è aperta;
- Visualizzare gli studenti iscritti ad una lista d'esame quando essa è stata chiusa;
- Eliminare dalla lista d'esame quegli studenti che pur essendosi iscritti non hanno sostenuto l'esame;
- Inserire i voti degli studenti che hanno sostenuto un certo esame;
- Pubblicare la lista degli studenti iscritti con allegati i voti conseguiti.

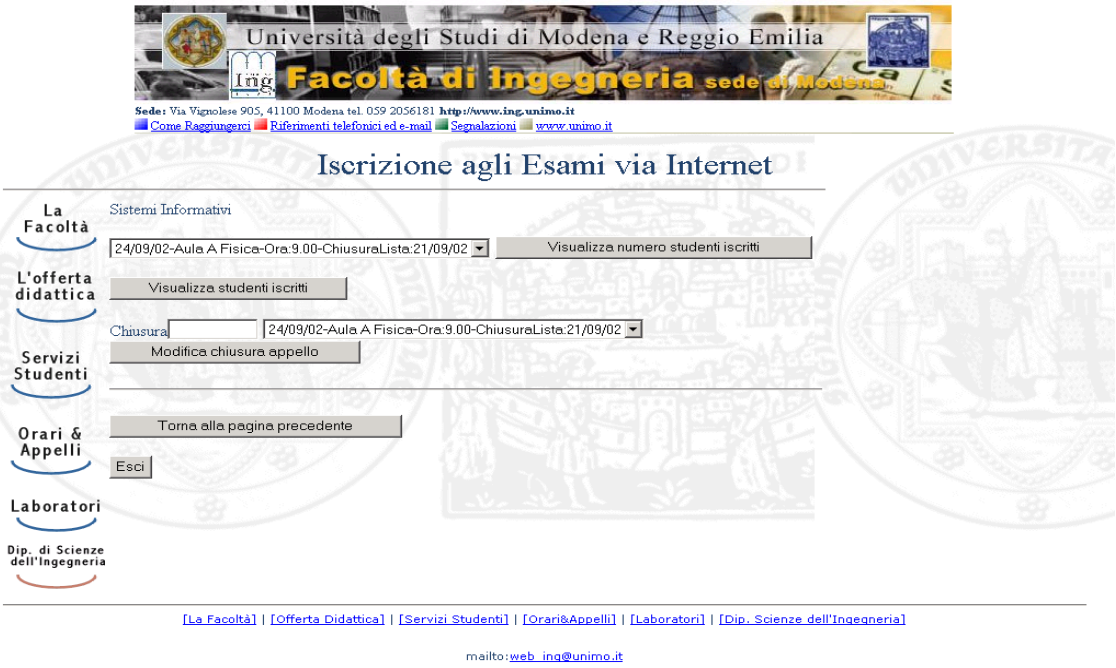


Figura A.10: Pagina di scelta-docente dell'operazione da eseguire

Come sottolineato dall'immagine seguente il docente può scegliere di lavorare su una lista ancora aperta o su una lista chiusa. Quello che è interessante notare è il fatto che viene mostrata l'elenco delle liste attualmente pubblicate.

Figura A.11: Pagina di scelta-docente della lista d'esame su cui lavorare

Nel caso in cui il docente decida di visualizzare gli studenti iscritti ad una lista quando ancora è aperta potrà soltanto visualizzare gli studenti iscritti.

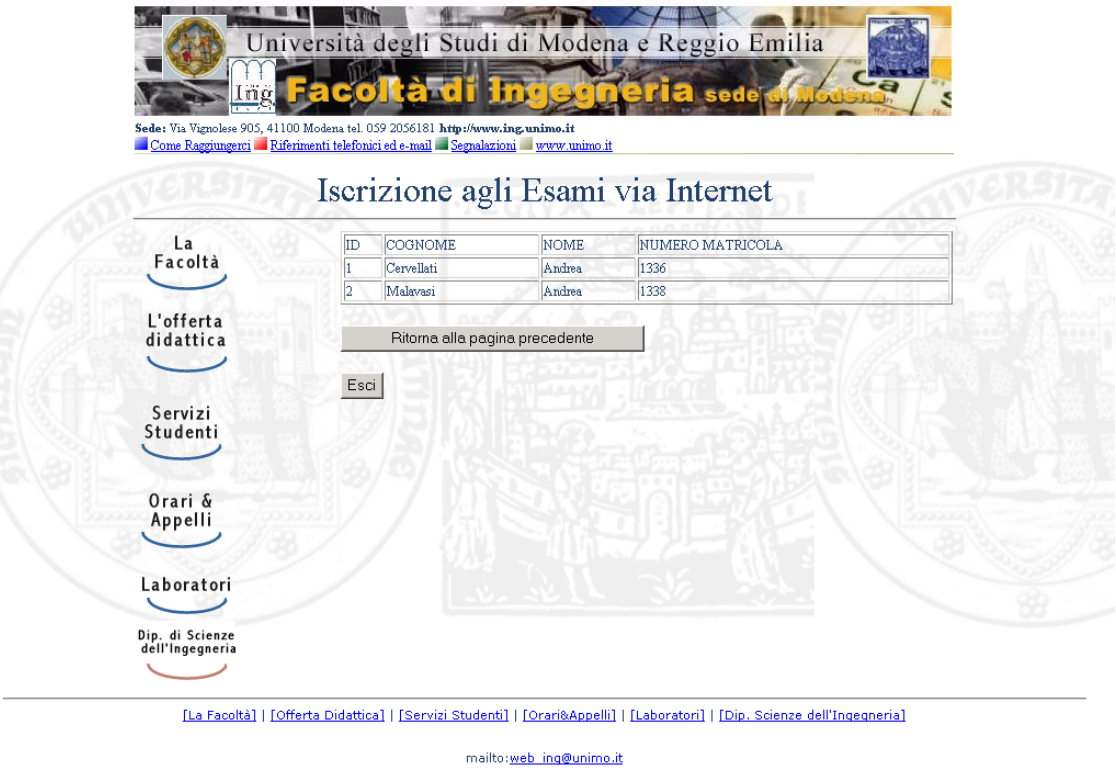


Figura A.12: Pagina di visualizzazione degli studenti iscritti ad un esame (lista aperta)

Nel caso in cui il docente decida di operare su una lista chiusa il docente potrà decidere di eliminare dalla lista tutti quegli studenti che pur essendosi iscritti non hanno sostenuto l'esame.

Figura A.13: Pagina di visualizzazione degli studenti iscritti ad un esame (lista chiusa)

Dopo aver compiuto questa operazione di scrematura il docente può procedere all'immissione dei voti degli studenti. Una volta conclusa questa operazione può decidere di mettere a disposizione degli studenti la lista degli iscritti con allegati i voti.

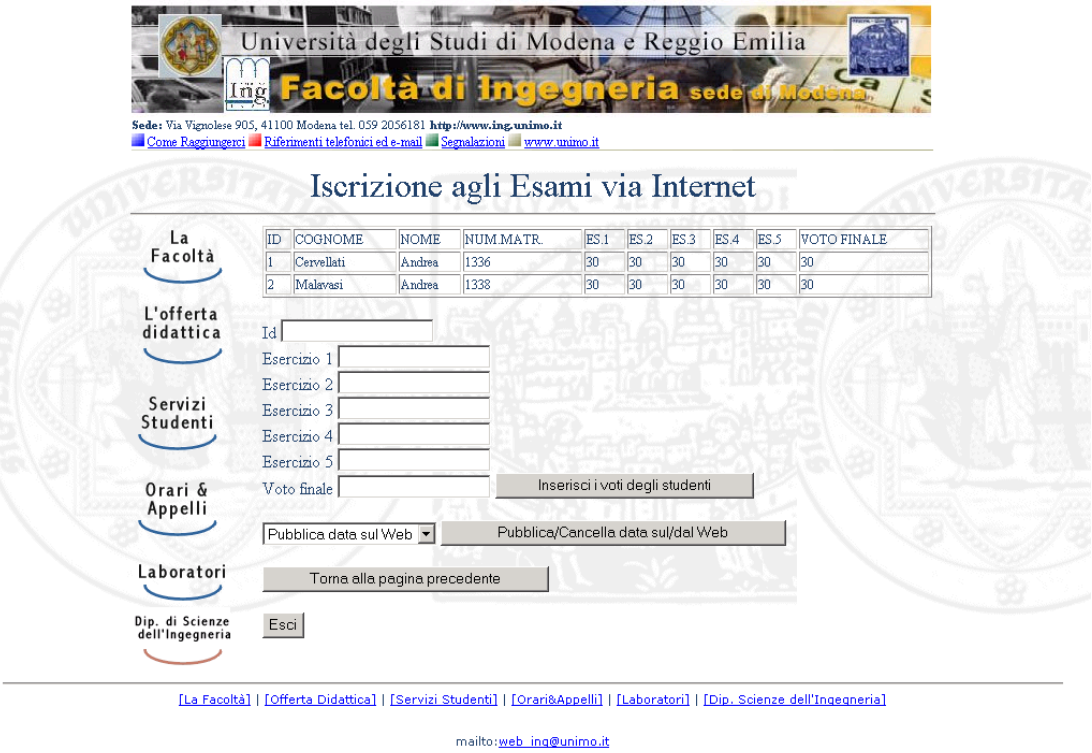


Figura A.14: Pagina di immissione e pubblicazione dei voti

Bibliografia

Bill Shannon
Java 2 Enteprise Edition specification
http://www.java.sun.com/

Eduardo Pelegrí-Lloport
Java Server Page specification
http://www.java.sun.com/

Linda G.DeMichiel, L.Umit Yalcinalp, Sanjeev Krishnon
Enterprise Java Bean specification
http://www.java.sun.com/

Kevin Boone, Tobias Frech, Scott Stark
JBoss documentation
http://www.jboss.org

Jakarta Tomcat documentation
http://jakarta.apache.org/tomcat/

Luca Ciocci
Portale Web di Facoltà: progetto e implementazione della funzione di gestione
di informazioni e materiale didattico a cura del docente
http://www.dbgroup.unimo.it/tesi/

Roberto Delfini
Un motore di news per portali Web:
progetto ed implementazione
http://www.dbgroup.unimo.it/tesi/

Martin Fower, Kendall Scott
UML distilled, quick reference to Object Modelling Language Standard
Addison-Wesley

Conclusioni

Come si è avuto modo di osservare dalla trattazione appena conclusa sono state affrontate tutte quante le problematiche connesse al progetto ed all'implementazione di un'applicazione server-side per la gestione degli appelli d'esame in ambito universitario.

La realizzazione di questo progetto ha richiesto la necessità di affrontare tutte quante le fasi che caratterizzano la produzione di un software completamente nuovo.

Un'ulteriore difficoltà è sorta dal fatto che quest'applicazione deve entrare a far parte del sito della facoltà di ingegneria e quindi deve essere integrata con applicazioni già esistenti . Infatti, nel

compiere le scelte di progetto è stato necessario tenere in considerazione tutte quante le scelte che erano già state prese in precedenza.

Si è cercato nel corso dei capitoli di mostrare i vari aspetti che hanno caratterizzato l'attuazione di quest'applicazione.

Dall'analisi dei requisiti richiesti per il compimento di questo progetto si è scelto di utilizzare una tecnologia, la J2EE (Java 2 Enterprise Edition), che oltre ad essere innovativa è affidabile e funzionale. Per realizzare questa applicazione è stato necessario ragionare considerando di organizzare l'applicazione in modo differente rispetto al passato. Infatti, nel momento in cui si è deciso di utilizzare la J2EE è stato inevitabile strutturare l'applicazione su tre livelli sviluppando separatamente le due parti che la caratterizzano: la business logic e presentation logic. Gli strumenti di base di cui si è fatto uso sono due componenti particolarmente efficienti ed attuali come gli EJB (Enterprise Java Bean) e le JSP (Java Server Page).

Concludendo si può dire che lo sviluppare questo progetto è stato particolarmente stimolante proprio perché ha permesso di affrontare tutti quanti gli aspetti che caratterizzano la creazione di qualcosa di nuovo: dalla scelta degli strumenti di cui far uso, al modo di soddisfare tutti quanti i requisiti richiesti fino all'utilizzo effettivo dei componenti scelti.