UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

FACOLTA' DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

Sviluppo di un'applicazione web per l'archiviazione e la visualizzazione di dati di espressione genica ottenuti con la tecnica del microarray.

RELATORE
PROF. ANTONIO D'ACIERNO

CANDIDATO
GIUSEPPE BATTINELLI
MATR. 041/3024

CORRELATORE
DOTT.SSA MARGHERITA MUTARELLI

ANNO ACCADEMICO 2005/2006

.....alla mia famiglia, che ha
sempre creduto in me, spronandomi
in ogni circostanza a dare il meglio
di me stesso,
senza la quale questa dedica non
sarebbe mai esistita….

Ringraziamenti.

Eccomi giunto, FINALMENTE, alla conclusione.

Cosa dire di questo lavoro?

Beh, potrei riassumerlo con 3 aggettivi:appassionante..
stimolante….STRESSANTE…

Volendolo riassumere con un pensiero, direi che sono soddisfatto ed orgoglioso di quanto realizzato e delle conoscenze acquisite durante questo lunghissimo percorso (con non poche difficoltà..).

Fosse questa una pagina dedicata all'applicazione che ho realizzato potrei anche concluderla qui, ma essendo questa la pagina dei "ringraziamenti" mi sa che, per la prima volta dall'inizio della scrittura di questo lavoro, dovrò cambiare tema e parlare d' altro.

Innanzitutto voglio ringraziare chi mi ha aiutato professionalmente a realizzare questo lavoro.

Parlo in prima persona dell'ing.Prof. Antonio d'Acierno, una persona come poche trovate nel mio percorso universitario, scherzosa e sincera quando c'era da esserlo, leale e pignola quando lo richiedevano le circostanze. Ha saputo sempre stimolarmi e valorizzarmi, dandomi sempre massima libertà di azione e di pensiero, "limitandosi" a guidarmi dalla sua notevole esperienza in materia.

Lo ringrazio perché in questi mesi non ha fatto altro che confermare, se non incrementare, la mia grande stima in lui acquisita durante i corsi universitari.

Un grazie immenso va alla dott.ssa Margherita Mutarelli, che si può dire in alcuni momenti abbia diviso con me il peso del lavoro, riuscendo con la sua simpatia e disponibilità ad aiutarmi a trovare la soluzione ad ogni problema che si presentava.

Non potrò dimenticare i pomeriggi passati nel tentativo di farmi capire come funzionasse un esperimento di microarray (non era facile con un cervello (il mio) abituato negli ultimi anni a capire solo numeri ed algoritmi…) ma alla fine credo proprio che ce l'abbia fatta..(?!?!)……..

Inoltre non ha mai smesso di essere presente nei miei confronti, anche in un momento particolare e impegnativo della sua carriera ed è per questo che le auguro le più grosse soddisfazioni professionali nella vita futura.

Un ringraziamento doveroso va anche al dott. Angelo Facchiano che ha coordinato il mio lavoro svolto con la dottoressa Mutarelli .

Finita la parentesi del CNR è ora di passare alla mia famiglia.

Il mio primo pensiero, ovviamente, va ai miei genitori, senza i quali non sarei mai potuto arrivare a questo punto e a questo traguardo.

Non parlo solo del sostegno economico, fondamentale in quanto mi ha consentito di "dormire" per anni a 5 minuti dall'università , permettendomi di guadagnare ore di studio e di riposo che avrei dovuto perdere viaggiando, che sicuramente è stato indispensabile.

Mi riferisco soprattutto a quell'aiuto morale a volte implicito, altre volte esplicito, che tante volte mi hanno mostrato in questi lunghissimi anni universitari.

Mi riferisco a tutte le occasioni in cui mia madre e mio padre, celando in silenzio l'ansia, mi hanno incoraggiato, vedendomi preso dai libri o preoccupato per un programma troppo duro, o per un esame scritto che sembrava proprio non volere andare (l'esame di Fisica, per esempio???), o per la mia a volte "invidia" per amici o conoscenti che avevano più tempo libero da dedicare a divertimenti e hobbies di quanto ne avessi io.

Ripenso a tutte le volte che mi hanno accompagnato a Napoli per evitarmi viaggi continui ed ogni tipo di fastidio che potesse distogliermi dallo studio, oppure i momenti in cui hanno dovuto sopportare i mie sbalzi d'umore e lo stress accumulato..potrei descrivere tanti momenti come questi, ma non c'è n'è bisogno, perché li porto tutti con me e per tutte queste cose posso solo ringraziarli.

Un ringraziamento a parte merita mio fratello Daniele, il quale mi ha saputo confortare nei pomeriggi di massima fatica e con il quale ho riso e scherzato nei momenti di massimo stress ed ansia e che mi ha aiutato nelle idee e nei consigli ogni volta che l'ho coinvolto nei miei problemi.

Anche lui è alla fine del primo percorso universitario e gli auguro di terminarlo presto e di avere ancora più soddisfazioni nell'ultimo round della vita universitaria che intravede all'orizzonte.

Durante il mio percorso universitario ho avuto la fortuna di incontrare e conoscere Paola, una ragazza che attualmente rappresenta davvero tanto per me, che dal primo giorno in cui è comparsa nella mia vita non ha fatto altro che spronarmi a tirare fuori il meglio di me, a volte anche esagerando nella stima e nella considerazione concessami.

La ringrazio per avermi saputo tirare su in alcuni momenti universitari non troppo felici e di avermi sopportato nel periodo finale…nonostante i suoi impegni e i suoi stress non fossero da meno.

Da sottolineare il lavoro svolto in collaborazione (inconsapevole ed a distanza) con la dott.ssa Mutarelli, per farmi riuscire ad entrare nel complicatissimo mondo della genetica (o meglio per far entrare la genetica nel mio cervello…).

E' per tutto questo che le auguro di tutto cuore di raggiungere le soddisfazioni ed i traguardi che merita, sperando di poterle ricambiare il conforto offertomi in questi periodi così delicati e stressanti e poterla presto festeggiare e lodare come tanto sta facendo in questi giorni con me.

Aspetto con ansia quel giorno, perché solo allora la mia festa sarà completa e il mio traguardo raggiunto.

Un saluto ed un ringraziamento va anche alla sua famiglia, che da quando mi ha conosciuto ha sempre mostrato stima nei miei confronti e sincera felicità nei momenti belli della mia vita universitaria.

Un grazie a parte va a Mario, che non ha esitato ad offrirmi la sua solita disponibilità nel momento finale del bisogno, aiutandomi nel "design" dell'applicazione.

Un ringraziamento a parte merita tutta la mia famiglia, zii e cugini, che mi hanno fatto sempre sentire una persona importante (fin troppo), attribuendomi a volte delle capacità magari eccessive, ma che hanno sempre fatto di tutto nei nostri "incontri" per tranquillizzare i miei stati d'animo e scherzare su episodi spiacevoli della mia carriera universitaria.

Li ringrazio tutti, perché insieme alla mia famiglia mi hanno fatto sentire in questi anni come un atleta che, impegnando ogni risorsa del suo corpo per tagliare il traguardo finale , trova forza e sostegno dal tifo che le persone gli offrono lungo il percorso.

Un ultimo ringraziamento (e non in ordine di importanza) va ai miei 2 veri compagni universitari, Vincenzo e Raffaele. Con loro ho instaurato un vero rapporto di amicizia che, partendo dalle scuole superiori, ci ha accompagnato fino ad oggi.

E' con loro che ho trascorso la mia permanenza a Napoli ed ho affrontato tutte le peripezie della mia carriera universitaria.

Oggi posso posso veramente dirlo che sono stati i miei veri ed unici compagni di "ventura" e di "sventura" (più sventura che ventura..) ed anche a loro auguro di terminare al più presto i loro studi per poterci riunire una sera (come anticipato circa 6/7 anni fa) e brindare all' università "che fu".

Spero di non aver dimenticato nessuno…

A tutti

Grazie di cuore

Indice

CAPITOLO 1–INTRODUZIONE

1. Premessa 2
2. Stato dell'arte. 3
3. Linee guida del progetto. 4
4. Passi fondamentali. 6
5. Introduzione agli strumenti utilizzati nella fase di progettazione ed implementazione dell'applicazione. 7
6. Architettura del sistema 9

CAPITOLO 2–MICROARRAY E FONDAMENTI DI GENETICA

1. Introduzione alla genetica e principali definizioni 11
   - 1.1. Organismi e cellule 11
   - 1.2. Le molecole delle cellule 12
   - 1.3. Cromosomi, geni e genomi 17
2. La tecnica del microarray 25
   - 2.1. Premessa 25
   - 2.2. L'esperimento microarray 27
   - 2.3. L'analisi dei dati: Image Quantify 30
   - 2.4. Algoritmi di normalizzazione e data minino 31
3. Descrizione di un esperimento : Il Miame 35
   - 3.1. Premessa 35
   - 3.2. La struttura del Miame 36

CAPITOLO 3–IL FRAMEWORK JSF

1. Perché la necessità di un framework? 39
   1.1. Definizione di framework 39
   1.2. Introduzione al framework Jsf 42
2. Design Patterns 44
   2.1. Pattern MVC 45
   2.2. Teamworking 50
3. Il framework JSF 51
4. Jsf nel mondo dei frameworks 70
5. Ciclo di vita di una pagina Java Server Face 73
   5.1. Scenari di elaborazione di una richiesta 74
   5.2. Ciclo di vita standard 75
6. Jsf Expression Language 80
7. Espandibilità del framework 81
   7.1. Creare un Custom converter 81
   7.2. Event Listener 83
   7.3. Custom Validator 85
   7.4. Custom UI Components 90
8. Gestione della sicurezza 101
9. Configurazione dell'applicazione 102
   9.1. Configurazione dei backing beans 103
   9.2. Localizzazione, internazionalizzazione e Messaggi 105
   9.3. Configurare le Navigation Rules 105

CAPITOLO 4 – PROGETTAZIONE DELL'APPLICAZIONE

1. Introduzione alla progettazione di un'applicazione web 108
    1.1. Software Requirements Specifications (SRS) 111
2. Class diagram 117
3. Use Cases Diagrams 120
    3.1 Generale 122
    3.2 Gestione utenti 124
    3.3 Gestione dati personali 125
    3.4 Ricerca 126
    3.5 Gestione fattori dell'esperimento 127
    3.6 Gestione piattaforme 128
    3.7 Rappresentazione testuale dei casi d'uso dell'applicazione 129
4. Activity Diagrams 148
    4.1. Activity diagram Generale 148
    4.2. Login 149
    4.3. Registrazione 151
    4.4. Richiesta username/password 152
    4.5. Modifica dati personali 153
    4.6. Logout 154
5. Sequence Diagrams 155
    5.1. Registrazione 156
    5.2. Modifica dati personali 157
    5.3. Richiesta username/password 158
    5.4. Logout 159
6. Progetto del database Microarray 160
    6.1. Costruzione del modello concettuale dei dati 161
    6.2. Dizionario dei dati 162
    6.3. Vincoli e valori di default 165
    6.4. Analisi delle ridondanze 165
    6.5. Costruzione del modello fisico dei dati 166

CAPITOLO 5 – IMPLEMENTAZIONE DELL'APPLICAZIONE MICROARRAY WEB

1. Perché un'applicazione web e non un sito web? 172
2. Tecnologie utilizzate nell'implementazione dell'applicazione web 174
3. L'applicazione MicroarrayWeb 190
   3.1. Introduzione 190
   3.2. Controller 191
      3.2.1. Gestione delle sessioni 192
      3.2.2. Protezione delle pagine 196
   3.3. View 197
      3.3.1. Le librerie standard 198
      3.3.2. I forms: contenuto e layout 198
      3.3.3. Costruzione condizionale dei contenuti 199
      3.3.4. Datatable 200
4. Screenshots dell'applicazione 202
5. Internazionalizzazione (I18N) 215
6. Validazione 217
   6.1. Custom validator 219
      6.1.1. Email validator 221
7. Connessioni al database e teoria del "pooling" 222
   7.1. Teoria del pooling 222
   7.2. Soluzione adottata nella web application 224
8. Inserimento sicuro dei dati e uso delle "transazioni" 228
9. Il framework Tiles 230
10. L'implementazione Myfaces 233
   10.1 Upload di un file 234
   10.2 Calendario 237
11. L'applet Vimida 237
12. Commons Email 239

CONCLUSIONI 241

APPENDICE A – Software Requirements Specification 243

APPENDICE B – Creazione ed inizializzazione del database 292

CAPITOLO UNO

INTRODUZIONE

1.1 Premessa
1.2 Stato dell'arte
1.3 Linee Guida del Progetto
1.4 Passi Fondamentali
1.5 Introduzione agli strumenti utilizzati nella fase di progettazione ed implementazione dell'applicazione.
1.6 Architettura del Sistema

Questo capitolo introduttivo illustra il problema affrontato e descrive gli obiettivi della tesi. Si procede ad una descrizione a grandi linee del contenuto dei capitoli successivi e dell'architettura del sistema. Il materiale presentato è dunque di fondamentale importanza per la comprensione di tutto il seguito del testo.

1.1 Premessa

Oggetto di questa tesi è la descrizione del processo di progettazione e realizzazione di una applicazione web per la memorizzazione, la gestione e la consultazione dei dati relativi ad esperimenti di espressione genica effettuati con la tecnica del microarray.

In particolare, data la complessità strutturale di un esperimento di tale tipo, abbiamo cercato di realizzare delle procedure che descrivano in maniera quanto più dettagliata ed esaustiva possibile tutte le fasi dell'esperimento, favorendo all'utente l'inserimento dei dati attraverso semplici ed intuitive pagine web.

Per fare ciò ci siamo serviti dello "standard de facto" per la descrizione del problema che è il Miame, giunto attualmente alla versione 1.1.

Parallelamente a questo abbiamo cercato di realizzare i più efficienti ed utili percorsi di ricerca, per garantire agli utenti di ricostruire a posteriori l'esperimento e ricavare in maniera semplice ed immediata tutti i dati di interesse di quest'ultimo.

Grazie all'integrazione nella nostra applicazione dell'applet Vimida abbiamo reso possibile la visualizzazione grafica dei dati normalizzati dell'esperimento, mediante le tecniche dell' Intelligenza Artificiale.

Vista l'importanza dei dati in questione abbiamo cercato di realizzare un'applicazione che garantisse un elevato grado di sicurezza nell'accesso, associando inoltre ad ogni esperimento un particolare grado di visibilità per impedirne la consultazione a personale addetto, ma non autorizzato.

La tesi tratta principalmente gli aspetti informatici del progetto, mentre vengono fornite solo le informazioni chimiche e biochimiche strettamente necessarie alla comprensione dello stesso.

1.2 Stato dell'arte

La necessità di archiviare l'enorme mole di dati che la moderna ricerca in biologia molecolare produce continuamente grazie al progresso tecnologico recente, spinge alla creazione, gestione e manutenzione di banche dati specializzate, dove a competenze di tipo strettamente informatico sono affiancate competenze di tipo biologico per l'archiviazione dei dati in modo corretto, ragionato e di facile reperibilità per la comunità scientifica.

Questa prima applicazione dell' informatica alla biologia ha dato origine ad una nuova disciplina, la bioinformatica.

Questa disciplina è caratterizzata dal fatto che le informazioni di tipo biologico da trattare sono diventate quantitativamente molto rilevanti e non è più possibile analizzarle senza l'ausilio di sistemi informatici; si pensi al progetto genoma umano che ha permesso l'identificazione dei circa 30000 geni dell'uomo ed è chiaro che solo con l'ausilio di un computer è possibile gestire efficientemente tutti questi dati.

Le applicazioni classiche della bioinformatica sono l'archiviazione, la ricerca, la predizione o l'analisi, della composizione o della struttura, delle molecole biologiche, come il DNA e le proteine.

Con il completamento del sequenziamento del genoma di molte specie sia vegetali che animali, tra cui quello dell'uomo, le applicazioni della bioinformatica si sono ampliate e diversificate.

Una tecnologia sviluppata a partire dalla conoscenza dei genomi è quella dei *microarray* , con la quale viene misurato contemporaneamente per un gran numero di geni il livello di espressione del gene, per vari tessuti ed in molteplici condizioni; questa tecnologia è di supporto ad un'altro ramo della bioinformatica, quello della *functional genomics* in cui si cerca di capire le funzioni e le interazioni dei geni e dei loro prodotti.

La functional genomics quindi punta alla scoperta delle funzioni biologiche dei geni e a scoprire le interazioni di insiemi di geni e dei loro prodotti (proteine,RNA) nelle cellule sane e in quelle malate e come questo è collegato alle funzioni dell'organismo che si sta analizzando.

L'applicazione che ci accingiamo a presentare si pone come obiettivo primario quello di mettere a disposizione informazioni utili alle persone coinvolte in studi genetici nell'ambito del microarray per la catalogazione e la consultazione di esperimenti realizzati con questa tecnica.

1.3 Linee Guida del Progetto

Nelle fasi iniziali del progetto fissiamo dei punti chiave che seguiremo nella realizzazione dell'applicazione. In particolare l'applicazione dovrà:

1. essere accessibile da due tipi di utenti, un amministratore e un utente generico;

2. essere accessibile via web tramite un'interfaccia amichevole, sia nella parte amministrativa privata che nella parte pubblica;

3. essere in grado di identificare l'amministratore e autorizzarlo ad accedere alla parte amministrativa privata, vietando l'accesso a chiunque altro;

4. garantire la sicurezza delle informazioni trattate nella parte amministrativa privata;

5. contenere un supporto multilingua, ovvero le informazioni devono essere consultabili in lingua inglese ed in lingua italiana, predisponendo l'eventuale aggiunta di ulteriori lingue in futuro;

6. essere facilmente gestibile e manutenibile.

Come si può vedere, queste semplici linee guida sono ben lungi dall'essere un rigoroso documento di specifiche: questo perché scopo di questo capitolo è fornire una panoramica di ciò che sarà presentato nei capitoli successivi.

Nell' appendice A sarà infatti mostrato un vero e proprio documento delle specifiche, realizzato seguendo lo standard IEEE-SRS-1998, mentre nel capitolo 4 saranno mostrati tutti i diagrammi Uml utili ad una migliore comprensione delle operazioni del sistema.

1.4 Passi Fondamentali

Nello svolgere il progetto abbiamo seguito gli steps riportati di seguito (ovviamente la presentazione in serie dei passi non è indicativa di un processo di sviluppo sequenziale):

1. Valutazione delle tecnologie utilizzate per la progettazione e realizzazione dell'applicazione web: questo step comprende l'analisi dei vari metodi di sviluppo di un prodotto software per il Web e lo studio delle varie tecnologie per la progettazione e realizzazione di applicazioni web, con l'obiettivo di selezionare la tecnologia che fosse più idonea alle nostre necessità.

2. Ricerca e selezione delle informazioni utili al progetto, con studio approfondito del dominio applicativo e degli standard per la descrizione delle informazioni.

3. Progettazione della base di dati: abbiamo analizzato le informazioni selezionate al fine di individuare quelle necessarie alla costruzione di un modello concettuale adatto all'applicazione da realizzare, per poi effettuare la trasformazione nel modello relazionale e quindi nel modello fisico che verrà utilizzato dall'applicazione.

4. Progettazione della applicazione web, in base alla metodologia progettuale selezionata.

5. Realizzazione della applicazione web, in base alla tecnologia scelta.

6. Testing dell'applicazione web, al fine di individuare il maggior numero possibile di malfunzionamenti e/o errori e correggerli.

7. Rilascio dell'applicazione e sua installazione sul server del C.N.R. di Avellino(Consiglio Nazionale delle Ricerche), dove è tuttora in uso.

1.5 Introduzione agli strumenti utilizzati nella fase di progettazione ed implementazione dell'applicazione.

La Web application che andremo ad analizzare è stata implementata con il framework JavaServer Faces; gli strumenti software utilizzati per lo sviluppo e gli ambienti di esecuzione sono i seguenti :

- Piattaforma *J2SE 5.0* (Java 2 Second Edition) , con uso delle tecnologie servlet e Jsp
- Ambiente IDE *Eclipse 3.2* con il plug-in *Exadel Studio* 4.0 per lo sviluppo grafico del file di configurazione di JavaServer Faces.
- Web Container/Server *Jakarta Apache Tomcat 5.5*;
- Sybase Power Designer 11
- Database Mysql 5.0

La piattaforma J2SE (Java2 Standard Edition) rappresenta la base per lo sviluppo e l'esecuzione delle applicazioni Java, in quanto mette a disposizione la JVM (Java Virtual Machine), oltre a tutte le classi principali e le librerie del linguaggio. Un'estensione di questa piattaforma è rappresentata dalla J2EE (Java2 Enterprise Edition) che fornisce il supporto per strumenti avanzati come i Web Services, gli EJB (Enterprise Java Beans) e JavaServer Faces.

In questo caso, non si è reso necessario l'utilizzo della piattaforma J2EE, in quanto la distribuzione del framework JSF può essere gestita anche semplicemente mediante le librerie JAR (Java ARchive) che contengono le classi dell'architettura.

Per l'ambiente di sviluppo IDE (Integrated Development Environment), la scelta è caduta sul progetto OpenSource Eclipse 3.2, dopo un rapido confronto con l'antagonista NetBeans.

Ha nettamente prevalso il primo, considerando il supporto fornito per lo sviluppo di applicazioni Web basate appunto su JSF, grazie al plug-in free di Exadel. Il secondo, invece, fino alla versione 4.1 non ha ancora fornito tale supporto, introdotto a partire dalla versione 5 in fase di Beta Testing.

Prendendo in esame tale versione, la semplicità ed il supporto offerti da Eclipse 3.2 sono comunque nettamente superiori.

La caratteristica principale di Eclipse è, appunto, quella di avere una architettura a plug-in, il che lo rende un ambiente molto versatile ed adatto alle varie esigenze. Il plug-in scelto per lo sviluppo dell'applicazione è **Exadel Studio** , giunto da poco alla versione 4.0 , il quale , oltre a supportare lo sviluppo di progetti con tecnologia Jsf, fornisce una versione dell'Application Server Tomcat (5.5), molto utile per testare velocemente l'applicazione in locale; ulteriori caratteristiche sono la possibilità di gestire graficamente la configurazione del progetto e la possibilità di utilizzare la scrittura assistita, per pagine JSP e XML.

Per la progettazione dell'applicazione Web è stato utilizzato il prodotto CASE (Computer Aided Software Engineering) **PowerDesigner 11,** uno strumento flessibile e con supporto per la stragrande maggioranza dei linguaggi di programmazione e dei DBMS presenti sul mercato. Tale prodotto, ottimo per la realizzazione del diagramma dei Casi d'Uso e per la progettazione del modello concettuale e fisico della base di dati, è stato utilizzato anche per la realizzazione dei diagrammi UML dell'applicazione in esame.

Per quanto riguarda il Web Container, la scelta è ovviamente caduta sul progetto OpenSource Jakarta Apache Tomcat, sicuramente tra i migliori per l'esecuzione di applicazioni Web realizzate in ambiente Java; in particolare abbiamo deciso di usufruire della versione 5.5 embedded in ExadelStudio.

E' altresì necessario specificare che per JavaServer Faces è stata scelta l'implementazione JSF-RI (Refernce Implementation) 1.1.0.1 e non MyFaces.

Infine per la creazione e gestione del database abbiamo deciso di puntare sul database Mysql, nella versione 5, in quanto ci offriva tutte le caratteristiche di cui necessitavamo, oltre al notevole vantaggio , come i software e gli strumenti precedentemente menzionati, di essere gratuito.

1.6 Architettura del Sistema

L'architettura del sistema è una architettura classica Client/Server (Fig. 1.1), caratterizzata dalla presenza di un server centrale (localizzato nell'Istituto di Scienze dell'Alimentazione (ISA) presso il Consiglio Nazionale delle Ricerche (CNR) di Avellino (*www.isa.cnr.it*)) e dalla presenza di due tipi di Client, un client per la gestione, inserimento e consultazione dei dati (amministratore) e un client per l'inserimento e la consultazione dei dati (utenti).

Ovviamente la suddivisione dei Client in due categorie è solo esemplificativa, in quanto, ad esempio, l'amministratore può essere anche un utente e quindi effettuare la consultazione dei dati stessi.

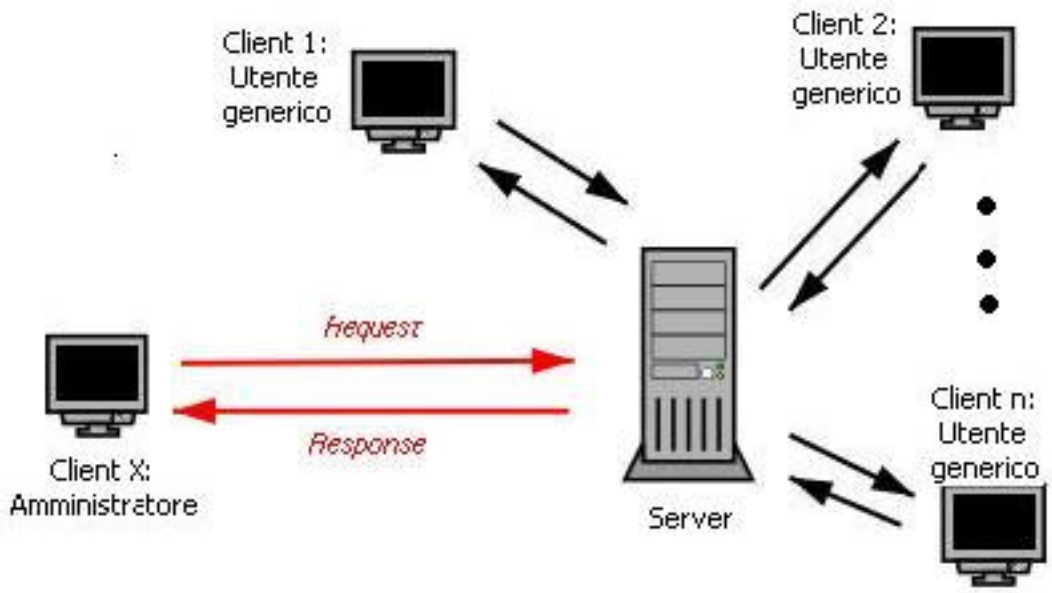


Figura 1.1 - ***Architettura Client/Server***

CAPITOLO DUE

MICROARRAY E FONDAMENTI DI GENETICA

2.1 Introduzione alla genetica e principali definizioni.
- 2.1.1 Organismi e cellule
- 2.1.2 Le molecole delle cellule
- 2.1.3 Cromosomi, geni e genomi

2.2 La tecnica del microarray
- 2.2.1 Premessa
- 2.2.2 L'esperimento microarray
- 2.2.3 L'analisi dei dati : Image quantify
- 2.2.4 Algoritmi di normalizzazione e di data mining

2.3 Descrizione di un esperimento:il MIAME
- 2.3.1 Premessa
- 2.3.2 La struttura del Miame

Questo capitolo si articola in 3 sottosezioni: la prima è incentrata nella descrizione dei concetti fondamentali della genetica, fornendo le definizioni dei principali termini che entreranno in gioco nella tecnica del microarray, la seconda si basa esclusivamente sulla descrizione della tecnica dei microarray ed infine la terza parte descrive il Miame, la sua struttura e importanza per la descrizione di un esperimento.

E' quindi un capitolo fondamentale perché rappresenta il dominio applicativo del nostro problema ed è ciò che andremo a progettare ed implementare nei capitoli successivi attraverso l'applicazione web in esame.

2.1 Introduzione alla genetica e principali definizioni

2.1.1 Organismi e cellule

Tutti gli organismi viventi sono composti da cellule.

Ogni cellula è un sistema complesso formato da molti elementi ed è racchiusa da una membrana. Gli organismi possono essere unicellulari e multicellulari; i batteri ed il lievito di birra sono esempi di organismi unicellulari, mentre un cane un gatto o un essere umano sono esempi di organismi multicellulari.

In un corpo umano si stima che siano presenti circa 6^{13} cellule, di circa 320 tipi differenti.

In natura ci sono due tipi differenti di organismi e di cellule rispettivamente: gli eucarioti (dal greco *eu* "buono" e *kàrion* "nucleo") ed i procarioti (dal greco *pro* "prima di" e *kàrion* "nucleo"). La distinzione tra procarioti ed eucarioti è piuttosto importante perché molti componenti costitutivi e processi vitali nei due tipi di cellule sono molto diversi. Si ritiene che queste differenze siano il risultato di differenti percorsi evolutivi.

Le cellule procariotiche hanno una struttura più semplice (ad esempio non hanno il nucleo) e sono più piccole (la grandezza tipica è di 1 micron) delle cellule eucariotiche, ma questo non significa che siano meno importanti. Le cellule eucariotiche hanno un nucleo, che è separato dal resto della cellula da una membrana, che contiene i cromosomi, che sono i portatori del materiale genetico. Ci sono inoltre altri elementi nella cellula racchiusi da membrane detti "organelli" che sono specializzati in svariati compiti.

I mitocondri in particolare sono specializzati nella produzione di energia (respirazione cellulare).

L'area della cellula esterna al nucleo e agli organelli è detta citoplasma. La membrana cellulare infine è una struttura complessa che ha una effettiva funzione di separazione della cellula dall'ambiente esterno, regola il flusso di cibo, energia ed informazioni da e verso la cellula.

Gli organismi multicellulari iniziano tipicamente la loro vita come cellula singola, risultato della fusione di una cellula sessuale maschile e di una femminile.

Questa cellula iniziale cresce, si divide e si diversifica in vari tipi di cellula, per formare i tessuti e, negli eucarioti superiori, gli organi.

Il processo di divisione e differenziazione è importante e delicato ed è controllato dalla cellula stessa. Malfunzionamenti in questo processo portano alla crescita incontrollata di cellule difettose e quindi ai tumori.

2.1.2 Le molecole delle cellule

Le cellule sono formate da molecole che sono state suddivise in quattro classi: le molecole piccole, le proteine, il DNA e l'RNA. Queste ultime tre sono conosciute anche con il nome di macromolecole biologiche.

Le molecole piccole.

Queste possono sia essere i blocchi costituenti delle macromolecole sia avere un ruolo indipendente, come essere un trasmettitore di segnali o una fonte di energia o materiale per le cellule. Qualche esempio importante ,oltre all'acqua, sono lo zucchero, gli acidi grassi, gli amminoacidi e i nucleotidi.

Un altro esempio sono gli amminoacidi che sono i blocchi costitutivi delle proteine; ci sono 20 differenti amminoacidi (ad essere più precisi ci sono 19 amminoacidi e uno che ha una struttura lievemente differente).

Le proteine.

Le proteine sono i principali blocchi costitutivi e funzionali della cellula e forniscono il 20% del peso di una cellula eucariotica, che è il contributo più alto dopo quello dell'acqua (70%). Ci sono molti tipi di proteine:

1. *le proteine strutturali*, che possono essere pensate come gli elementi costitutivi base degli organismi,(un esempio è il collagene, che è la principale proteina strutturale del tessuto connettivo e delle ossa).
2. *gli enzimi* ,che permettono una moltitudine di reazioni biochimiche, come l'alterazione, l'unione o la divisione delle altre molecole. L'insieme delle reazioni e dei percorsi che fanno gli enzimi è chiamato metabolismo.

3. *le proteine della membrana cellulare* che sono la chiave per la manutenzione dell'ambiente cellulare, regolando il volume delle cellule, l'estrazione o la concentrazione di piccole molecole dall'ambiente extracellulare.

Le proteine sono lunghe catene di amminoacidi ma ,a causa delle attrazioni e repulsioni tra gli atomi di questi amminoacidi, hanno una complessa struttura tridimensionale (vedi le figure seguenti), e per un'analisi più accurata questa struttura è stata suddivisa in quattro livelli.

1. Il primo livello è detto *struttura primaria* ed è una struttura lineare in cui le proteine sono rappresentate dalla successione dei 20 simboli degli amminoacidi che le compongono. Tale struttura è detta catena polipeptidica. La lunghezza delle proteine può variare da pochi a molte migliaia di amminoacidi componenti.
2. Le *strutture secondarie* dipendono dalla sequenza degli amminoacidi e sono le eliche alfa , i filamenti beta e strutture meno regolari dette loops. Anche le strutture secondarie vengono in contatto tra loro e anche tra queste si stabilisce un equilibrio di forze intermolecolari che ne determina una struttura compatta e relativamente stabile , detta
3. *struttura terziaria*.

Una proteina può essere formata da più di una catena di amminoacidi, in questo caso si dice che abbiamo una

4. *struttura quaternaria*.

Le proteine sono troppo piccole per poter essere viste in un microscopio ottico e scoprire la loro struttura è difficile e costoso.

Le immagini seguenti mostrano la struttura di una proteina usando particolari programmi di visualizzazione 3D, le immagini sono solo un modello della proteina, non 'fotografie', in quanto per le dimensioni, movimento e il principio di indeterminazione di Heisenberg non sarebbero possibili.

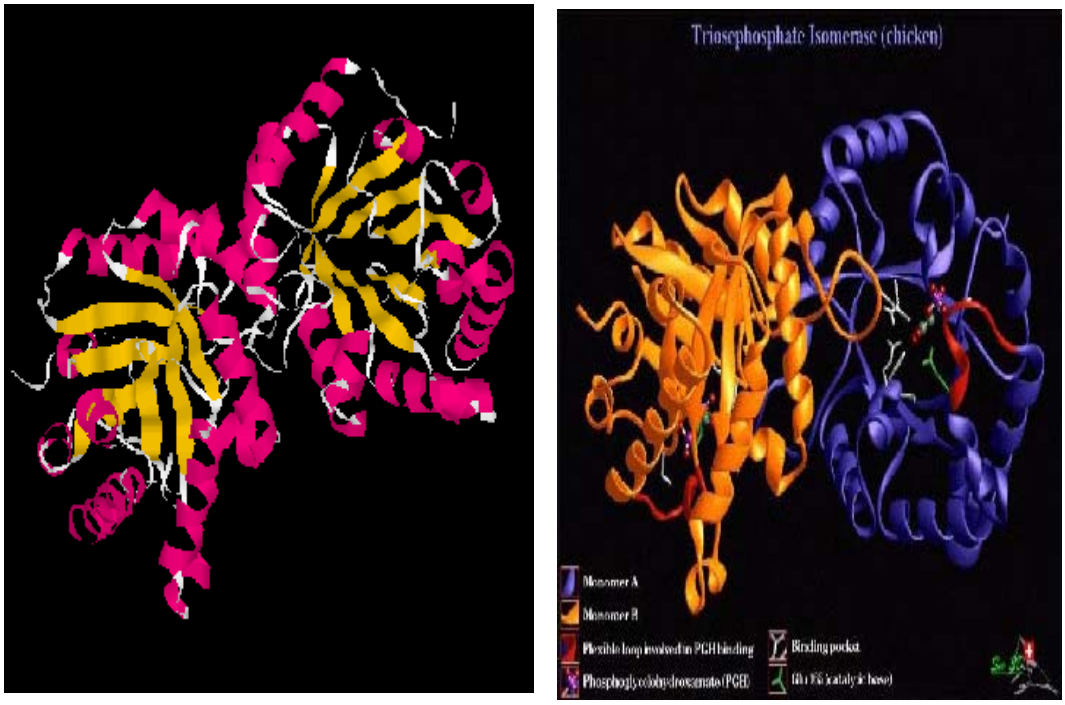


Figura 2.1 - ***modello di proteina (a sinistra), il colore magenta è per le eliche alfa, il colore giallo per i filamenti beta; (a destra) altro esempio di proteina in cui i colori evidenziano monomeri diversi***

Tutti e quattro i livelli strutturali delle proteine sono essenzialmente determinati dal primo livello, cioè dalla sequenza di amminoacidi e dall'ambiente fisicochimico in cui la molecola è posta. Predire la struttura della proteina a partire dalla sequenza degli amminoacidi è una delle sfide più importanti per la bioinformatica.

Il DNA.

Il DNA è la principale molecola portatrice di informazioni in una cellula. Il DNA può essere a filamento singolo o doppio. Una molecola a singolo filamento è anche chiamata polinucleotide ed è una catena di molecole più piccole chiamate nucleotidi.

Ci sono quattro tipi differenti di nucleotidi in due gruppi distinti:

1. Le *purine*
   - adenina
   - guanina

2. Le *pirimidine*
   - citosina
   - timina

Queste sono chiamate di solito basi (infatti le basi sono gli unici elementi distinguibili tra differenti nucleotidi) e denotati dalle loro lettere iniziali A,C,G e T (da non confondersi con gli amminoacidi).

Due filamenti complementari formano una struttura stabile, che assomiglia ad un'elica ed è conosciuta come la doppia elica del DNA.

Sono necessarie circa 10 coppie base per un intero giro che è lungo circa 3.4 nm.

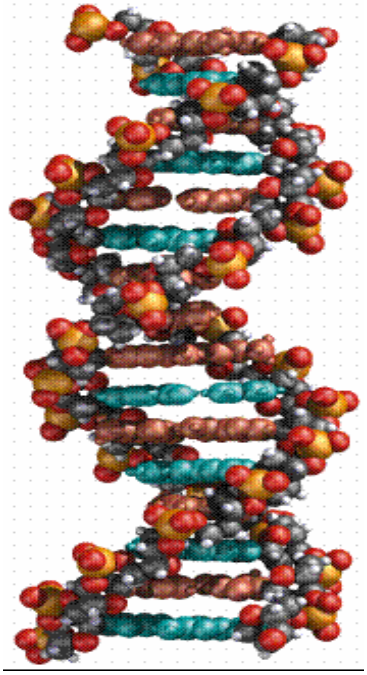


Figura 2.2 – ***Struttura tridimensionale a doppia elica del DNA (sopra). James Watson e Francis Crik ed il loro modello di DNA (sotto). (figure tratte da "On-Line Biology Book")***

Questa struttura fu mostrata per la prima volta a Cambridge nel 1953 da James Watson e Francis Crik (con l'aiuto di altri) e fruttò loro la vincita del premio Nobel. Due filamenti complementari di DNA formano una struttura stabile quasi indipendentemente dalla sequenza di nucleotidi che li formano.

Dato che la lunghezza tra due nucleotidi è di 0.34 nm, la densità lineare di immagazzinamento delle informazioni è di 68 bits/cm che è pari a circa 75 GB/cm. La complementarietà dei due filamenti è usata per copiare le molecole di DNA in un processo conosciuto come 'replicazione del DNA' ,in cui una doppia elica di DNA è replicata in una coppia di molecole uguali: la doppia elica del DNA durante il processo si biforca e un nuovo filamento complementare è sintetizzato su ognuna delle parti per avere alla fine del processo due eliche identiche all'originale.

In una cellula questo accade durante la divisione cellulare e una copia identica all'originale viene distribuita in ogni cellula.

Se la somma totale delle forze dei legami tra i nucleotidi complementari è abbastanza intensa da conservare la struttura, sono possibili anche nucleotidi spaiati tra i filamenti.

Il DNA che non è più in forma elicoidale viene detto *denaturato*.

L' RNA.

L'RNA ,come il DNA, è costituito da nucleotidi, ma presenta l'uracile U al posto della timina T. A causa di questa differenza l' RNA non forma una doppia elica, ma un singolo filamento .

L'RNA ha varie funzioni nella cellula, alcune di queste discusse nella sezione successiva, ad esempio, l'mRNA e il tRNA sono due tipi di RNA funzionalmente differenti ma che servono entrambi per la sintesi di proteine.

L'RNA può legarsi in modo complementare ad una singola stringa di DNA dove T è sostituito da U.

L'RNA può codificare le informazioni genetiche.

2.1.3 Cromosomi, geni e genomi

In una cellula ci sono una o più molecole di DNA ad elica organizzate in cromosomi. Negli Eucarioti i cromosomi hanno una struttura complessa e il DNA gira intorno a proteine strutturali chiamate istoni.

Un essere umano ha 23 coppie di cromosomi, che sono grandi abbastanza da poter essere viste con un microscopio ottico. La lunghezza totale del DNA in una cellula umana, se potessimo svolgerla, sarebbe più di un metro.

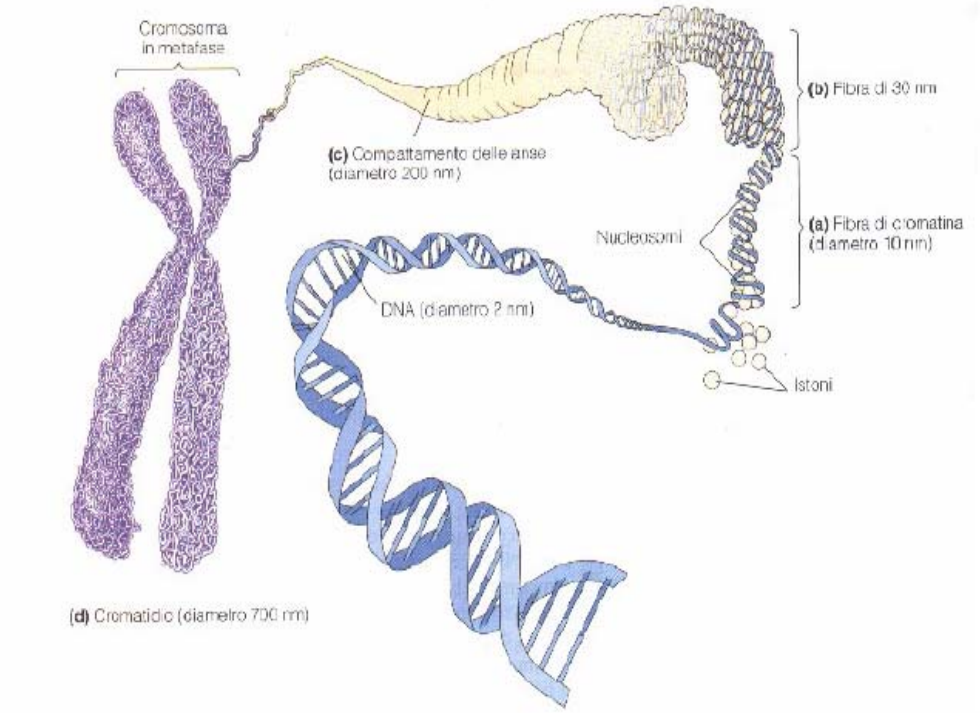


Figura 2.3 – ***tappe del compattamento dei cromosomi eucariotici (figura tratta da "Genetica,analisi e principi")***

Anche i mitocondri contengono DNA, ma questa quantità è minuscola rispetto a quella dei cromosomi. Il DNA dei cromosomi e dei mitocondri forma il genoma dell'organismo. Tutti gli organismi hanno il genoma e si ritiene che quasi tutte le informazioni ereditarie dell'organismo siano codificate in esso. Negli eucarioti i cromosomi sono nel nucleo.

Tutte le cellule di un organismo contengono il medesimo patrimonio genetico. C'è un meccanismo molecolare nelle cellule che mantiene intatto il DNA in entrambe le stringhe: se una stringa è danneggiata, è riparata usando l'altra come riferimento. E' importante che un danno al DNA, causato ad esempio da cause esterne come le radiazioni, come la rottura di una o entrambi i filamenti o il disaccoppiamento delle basi, porti soprattutto all'interruzione del meccanismo di replicazione del DNA.

Se il danno al DNA non è riparato si hanno due alternative, la morte della cellula o il tumore. I cambiamenti nel DNA sono dette mutazioni.

La dimensione del genoma differisce abbastanza considerevolmente da organismo ad organismo.

La determinazione della sequenza delle quattro lettere per un dato DNA è detto s*equenziamento* del DNA.

Il primo genoma intero di un batterio fu individuato nel 1995. Il genoma del lievito nel 1997, il verme nel 1999, la mosca nel 2000, l'erba nel 2001, e nel 2001 più del 90% del genoma umano è stato individuato.

L'individuazione dei genomi di batteri è diventata routine ed è fatta per la maggior parte da robot e completata da ricercatori umani. Il problema principale è la minimizzazione dei costi per lettera e la massimizzazione della velocità mantenendo la qualità. L'individuazione di genomi di dimensioni rilevanti come quello umano è ancora difficile, sebbene i maggiori problemi siano di tipo computazionale.

I robot sono in grado di individuare solo piccole sequenze alla volta che devono poi essere assemblate con l'utilizzo di algoritmi di assemblaggio. La maggiore difficoltà è che il genoma degli eucarioti superiori come l'uomo ha molte sottosequenze ripetute che rendono l'assemblaggio complicato e questo comporta un notevole intervento umano nel progetto di sequenziamento. I genomi contengono geni, la maggior parte dei quali codifica proteine.

I geni e la sintesi delle proteine.

Ci sono molte discussioni tra i biologi per trovare una definizione di gene che sia esauriente per tutti. Per i nostri scopi possiamo proporre questa definizione: *"Un gene è una stringa continua di DNA, dalla quale un complesso meccanismo molecolare può leggere informazioni (codificate come una stringa di A, T, G e C) e formare un particolare tipo di proteina oppure poche proteine differenti"*.

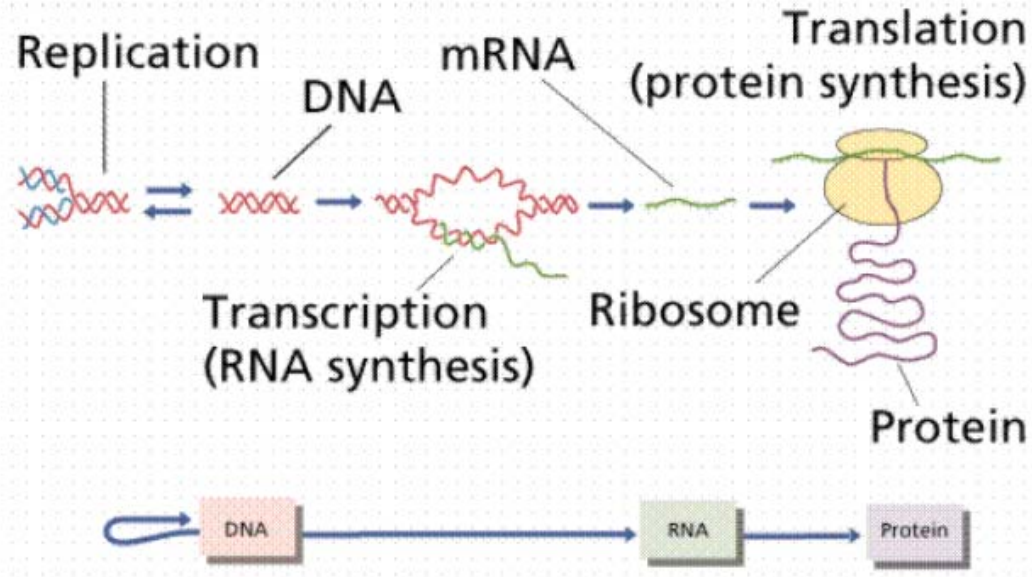


Figura 2.4 – ***processo di sintesi delle proteine***

Questa definizione non è precisa e per capirla meglio abbiamo bisogno di descrivere il meccanismo molecolare di creazione delle proteine basato sulle informazioni codificate nei geni. Questo processo è chiamato sintesi proteica ed ha tre fasi essenziali: trascrizione, splicing e traduzione.

Nella fase di trascrizione un filamento di DNA è copiato in un pre-mRNA (pre per preliminare e m per messaggero) dal complesso proteico RNA polimerase .

Nel processo i due filamenti di DNA legati in doppia elica vengono slegati e l'informazione è letta solo da un filamento .

Nella fase di splicing ("taglio e saldatura") alcuni tratti di pre-mRNA, chiamati introni, vengono rimossi e le sezioni rimanenti, chiamate esoni, vengono unite. La rimozione degli introni è una conseguenza della modalità con la quale i genomi degli eucarioti sono organizzati. Il DNA genomico che corrisponde al codice del gene non è continuo, ma consiste di introni ed esoni. Gli esoni sono la parte del gene che codifica le proteine e sono interspaziati da introni che devono essere rimossi nella fase di splicing.

Il numero e le dimensioni degli introni ed esoni differisce considerevolmente tra un gene ed un altro e anche tra specie diverse.

Solo alcuni geni nel lievito hanno gli introni, mentre negli uomini ci sono in media circa 4 introni per gene, e la lunghezza media di un esone è di 150 bp e circa 3400 bp per un introne.

I geni dei procarioti non hanno introni e la fase di splicing non è presente. Il risultato della fase di splicing è l' mRMA. Molti geni degli eucarioti hanno differenti varianti della fase di splicing, dove ad esempio alcuni premRNA producono differenti mRNA.

Questo processo è detto splicing alternativo.

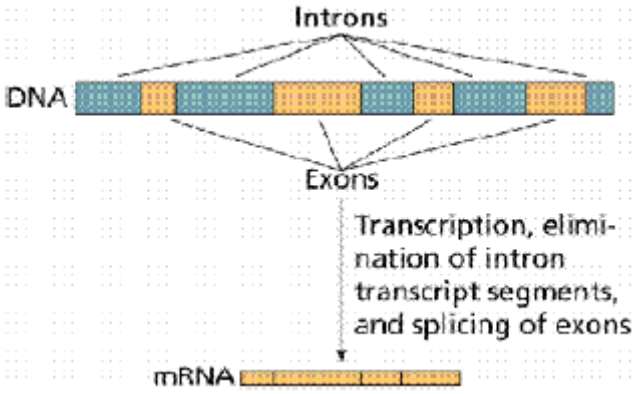


Figura 2.5 – ***fase di splicing (figura tratta da On-Line Biology Book)***

La fase di traduzione è il processo di costruzione delle proteine attraverso l'unione di amminoacidi nell'ordine dato dal codice dell'mRNA. L'ordine degli amminoacidi è determinato da 3 nucleotidi adiacenti nel DNA. Ogni tripletta è chiamata codone ed è il codice di un solo amminoacido.

Il prodotto finale della traduzione è la proteina.

La predizione e conteggio dei geni.

Una domanda interessante: data una sequenza di DNA genomico, possiamo dire dove si trova il gene nel cromosoma? La risposta è che possiamo, sebbene l'accuratezza di una tale predizione non sia molto alta. La maggior parte della conoscenza di queste predizioni viene da geni identificati sperimentalmente.

Ci sono circa 6700 geni sperimentalmente confermati nel genoma umano (per essere più accurati 6700 cDNA verificati). La predizione dei geni è un problema importante per la bioinformatica e ci sono vari algoritmi che tentano questa predizione usando i geni conosciuti come dati di partenza. Una tecnica algoritmica popolare usata nella predizione di geni è quella dei modelli nascosti di Markov (HMM). La predizione automatica di geni è stata importante nel dare senso ai dati non completi o non terminati dei vari progetti genoma: la sequenza totale è assemblata a partire da marcatori genetici conosciuti sui vari cromosomi.

Anche se conoscessimo dove i geni sono nel genoma, non è ovvio come contarli.

A causa dell'esistenza di geni sovrapposti e delle varianti, è difficile definire quale parte del DNA possa essere legato allo stesso o a differenti geni.

Tuttavia, anche se in modo approssimato , un conteggio dei geni in un organismo si è riusciti a farlo ed alcuni risultati hanno dato delle sorprese.

Organismo	Numero di geni predetti	Parte del genoma che codificante (esoni)
E.Coli (bacterio)	5000	90%
lievito	6000	70%
verme	18000	27%
mosca	14000	20%
erba	25500	20%
uomo	30000	< 5%

Figura 2.6 – ***confronto geni predetti tra vari organismi***

Una delle sorprese è il numero relativamente basso di geni in un genoma umano rispetto a quello di un verme. La ragione del basso numero di geni nel genoma umano potrebbe essere l'elevato numero di varianti che i geni umani hanno nella sintesi proteica, ma questo non è stato ancora provato.

La presenza del 95% di DNA non codificante nel genoma umano rimane un mistero. Ci sono molte ipotesi che cercano di spiegare il fenomeno, ma nessuna è generalmente accettata.

La somiglianza dei genomi, SNPs e il confronto di genomi.

Si ritiene che i genomi dei vari individui siano equivalenti al 99,9% ed in media solo un nucleotide su mille è differente nei genomi di due diversi individui.

Quindi possiamo parlare in termini di genoma umano in generale.

Quello 0.1% di differenze è usato per il riconoscimento degli individui: si analizzano le variazioni in parti non codificanti del genoma e si producono schemi che possono distinguere in modo affidabile due individui differenti. Nonostante questo si possono avere dei problemi per la distinzione di due gemelli identici.

Variazioni particolarmente importanti nei genomi individuali sono i *singles nucleotide polymorphisms* o SNPs, che possono presentarsi sia nelle parti codificanti che in quelle non codificanti del genoma.

Le SNPs sono variazioni di sequenze di DNA che si presentano quando una singola base (A,C,T,G) è alterata cosicchè differenti individui possono avere lettere differenti in posizioni isotope. Particolari nucleotidi nelle posizioni SNP dentro o vicino ai geni possono influenzare la proteina prodotta dal gene.

Bioinformatica e functional genomics

Abbiamo accennato nell'introduzione alla nascita e allo sviluppo rapidissimo che sta avendo la bioinformatica.

Le applicazioni classiche della bioinformatica sono l'archiviazione, la ricerca, la predizione o l'analisi, della composizione o della struttura, delle molecole biologiche, come il DNA e le proteine.

Con il completamento del sequenziamento del genoma di molte specie sia vegetali che animali, tra cui quello dell'uomo, le applicazioni della bioinformatica si sono ampliate e diversificate.

Una prima applicazione è quella di confrontare i genomi delle varie specie per osservarne differenze e similitudini e per studiarne l'evoluzione; questo ramo della bioinformatica è detto *comparative genomics*.

Un'altra tecnologia sviluppata a partire dalla conoscenza dei genomi è quella dei *microarray*, in cui viene misurato contemporaneamente per un gran numero di geni il livello di espressione del gene, per vari tessuti ed in molteplici condizioni; questa tecnologia è di supporto ad un'altro ramo della bioinformatica, quello della *functional genomics* in cui si cerca di capire le funzioni e le interazioni dei geni e dei loro prodotti.

2.2 La tecnica del microarray

2.2.1 Premessa

Il progetto Genoma Umano e progetti simili finalizzati al sequenziamento degli altri genomi hanno aperto una nuova era per la ricerca biomedica, la cosìdetta era “post genoma”.

Con la conoscenza dell’intera sequenza del genoma e la sua analisi finalizzata all’identificazione di nuovi geni, la sfida per i ricercatori è la comprensione di come, quando e perché i geni si attivano, così come, a livello di proteina, quando certe proteine si sintetizzano o modificano, e come funzionano all’interno della cellula.

La possibilità di analizzare nello stesso tempo il livello di espressione di migliaia di geni o proteine è una grande potenzialità per la ricerca medica e biologica, nonché la caratteristica e il pregio peculiare della tecnica del microarray.

Per esempio è possibile analizzare come l’espressione di migliaia di geni è modificata in una cellula tumorale sotto uno specifico trattamento farmacologico, con la possibilità di valutare differenti trattamenti a livello genetico e quindi l’eventuale efficacia del trattamento stesso.

La tecnica dei microarray nasce negli anni 90 al fine di indagare i livelli di espressione dei geni in una cellula. Essa è in grado di stabilire se un certo gene è attivo in quella cellula, cioè se la proteina corrispondente viene effettivamente prodotta e in quale misura. Oggi è possibile stabilire l’espressione per migliaia di geni contemporaneamente, confrontando allo stesso tempo due diversi tessuti, ad esempio campioni provenienti da pazienti diversi (uno sano e uno ammalato) o dallo stesso paziente (prima e dopo la cura).

La tecnologia microarray è composta da due parti fondamentali, una parte hardware, formata da macchinari che preparano i vetrini che sono di supporto all'esperimento, da laser e scanner, che rilevano i risultati dell'esperimento e da una parte software formata da programmi che analizzano i risultati dell'esperimento.

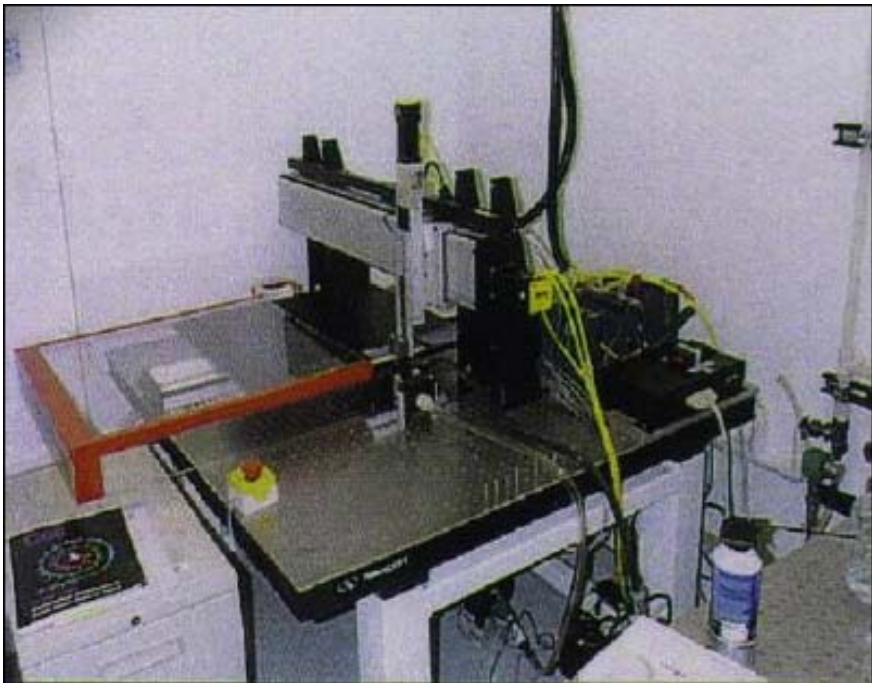


Figura 2.7 – ***l'arrayer***

Vedremo in seguito in cosa consiste un esperimento microarray e che tipo di analisi bisogna fare sui dati per ottenere informazioni utili ai biologi.

E' importante ribadire che questa tecnologia si basa sulla conoscenza dei geni del DNA degli organismi sottoposti ad analisi e quindi è utilizzabile solo per organismi di cui si conosce il genoma (al momento solo di alcuni organismi vegetali, animali e in particolare quello dell'uomo).

2.2.2 L’esperimento microarray

Come accennato nell’introduzione, un esperimento microarray mette a confronto l’espressione dei geni di due tipi di cellule, un tipo utilizzato come riferimento ed un tipo da sottoporre a test. Un esperimento quindi inizia con la fase di scelta delle cellule da sottoporre ad analisi.

Le scelte più probabili potrebbero essere:

- cellule di due tessuti differenti, cosi da rilevare le differenze tra i geni attivi in un tessuto rispetto a quelli di un altro tessuto.
- cellule appartenenti allo stesso tessuto ma di due esemplari differenti ,uno sano ed uno malato.
- cellule dello stesso tessuto ma sottoposte a due tipi di ambiente diverso, differenti condizioni di temperatura o di alimentazione.
- cellule dello stesso tessuto ma prelevati ad intervalli di tempo differenti per analizzare il ciclo cellulare.

Queste sono le strategie più frequenti di scelta, ma potrebbero essere selezionati criteri ancora diversi per la scelta delle cellule.

Dopo questa fase si procede all’esperimento vero e proprio illustrato nella figura seguente.

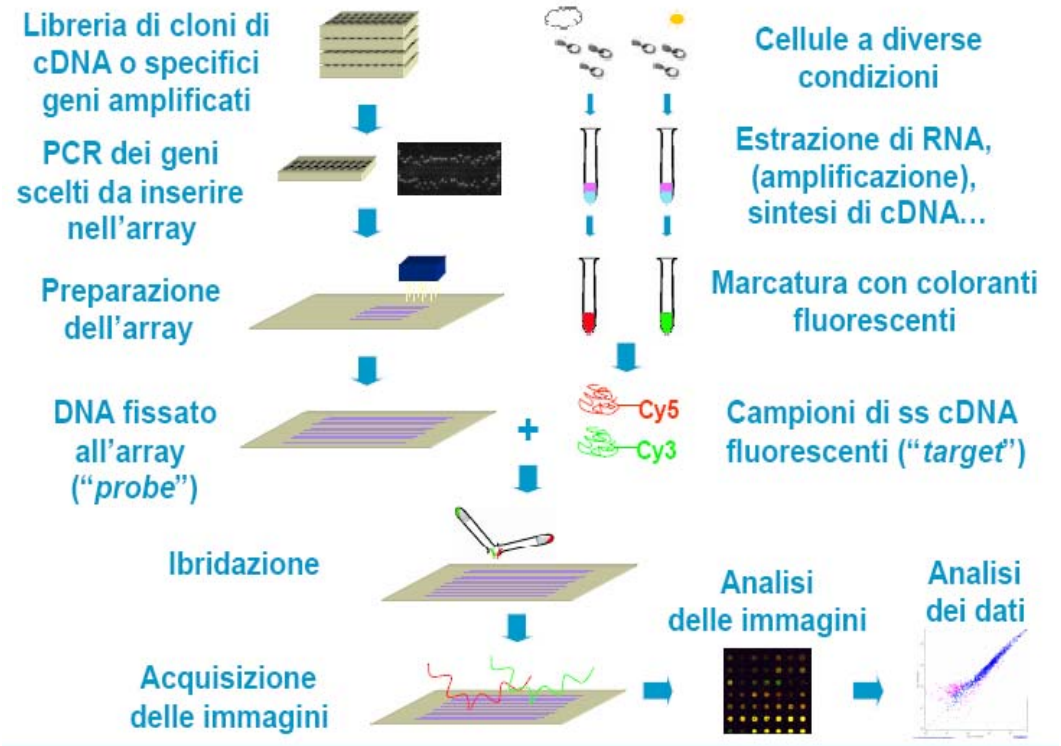


Figura 2.8 – ***Processo di realizzazione di un esperimento di Microarray***

Un microarray, o DNA-chip, è un sistema miniaturizzato in cui su un apposito supporto di pochi cm^2 vengono immobilizzati acidi nucleici a sequenza nota, detti sonde o probes, corrispondenti ai geni che si vogliono studiare.

Da un campione biologico (la cellula in esame) si preleva un'opportuna quantità di mRNA (Rna messaggero) , che è il prodotto intermedio del processo che dai geni conduce alle proteine e che ha la caratteristica di riconoscere le sonde complementari alle quali si lega chimicamente.

L'mRNA indica quali e quanti geni sono espressi in quell'istante; l'mRNA è un acido che si degrada velocemente, quindi ne viene fatta una trascrizione inversa (reverse transcription) così da ottenere un DNA complementare (cDNA).

Quest'ultimo viene colorato con elementi fluorescenti che si legano alle molecole di cDNA e che reagiscono solo alla luce di una specifica lunghezza d'onda, così da poter essere rilevati con laser di lunghezza d'onda opportuna.

Finita questa fase inizia la fase di ibridazione: il cDNA ottenuto dai due tipi differenti viene posto in un unico recipiente e poi viene messo a contatto con un vetrino microarray precedentemente preparato.

Sul vetrino sono state poste ordinatamente, con una struttura a griglia, piccole gocce di DNA,ognuna di uno specifico gene da analizzare.

Il vetrino è messo in contatto con la miscela dei due cDNA colorati e il DNA del microarray si legherà solo al cDNA dello stesso gene e i colori saranno utilizzati per determinare i livelli di quel gene nei due tipi differenti.

Dopo la fase di ibridazione viene quella di scanning in cui laser a lunghezza d'onda differenti colpiscono il vetrino microarray e tramite un sensore viene rilevata un'immagine del vetrino, una per ognuno dei due colori.

Queste immagini sono in bianco e nero e i colori visualizzati sono solo elaborazioni al computer: alla prima immagine si attribuisce un colore verde, alla seconda un colore rosso ed unendo le due immagini si ottiene una gradazione di colori dal verde al rosso passando per il giallo. Il colore in questo caso indica i rapporti tra i livelli dei geni nei due tessuti.

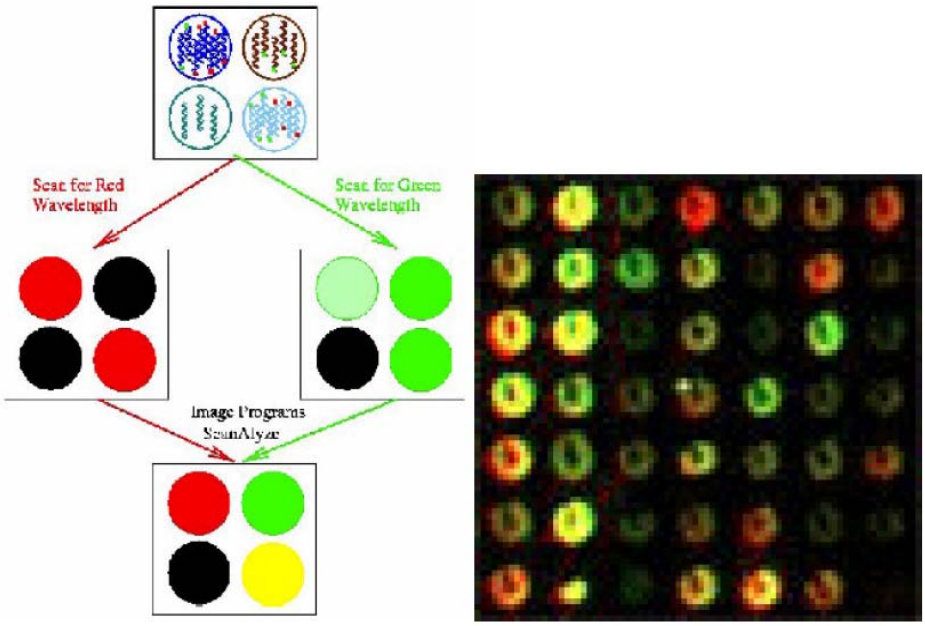


Figura 2.9 – ***Significato dei colori e sovrapposizione delle 2 foto***

Ottenuta l'immagine si passa dalla parte hardware a quella software e si inizia la fase di analisi dell'immagine, che potrebbe essere riassunta da questa figura:

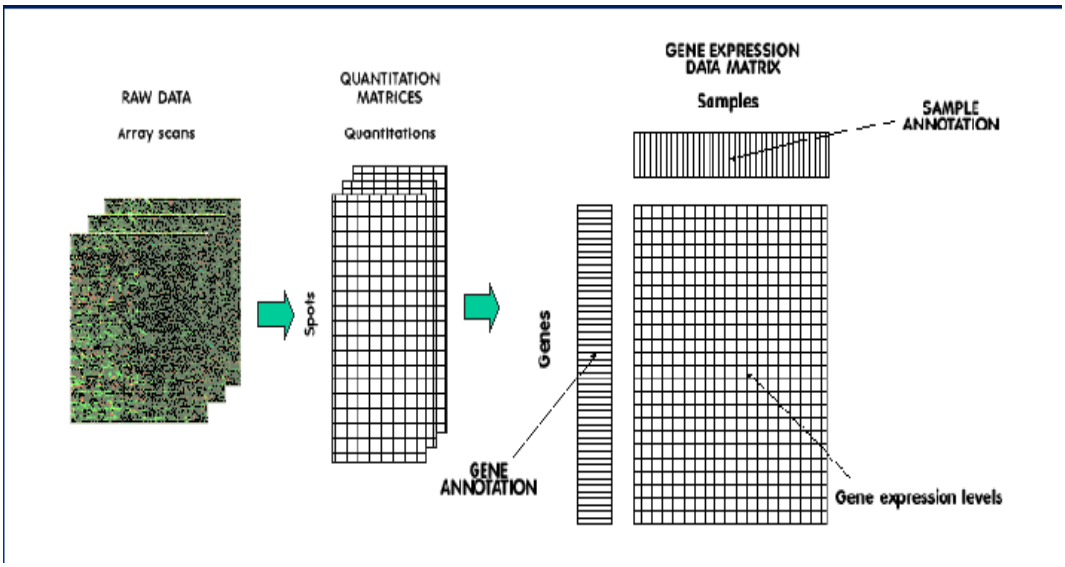


Figura 2.10 – ***Processo di analisi dei dati a partire dalle foto dell'array***

2.2.3 L'analisi dei dati : Image quantify

Le immagini ottenute dall'esperimento sono sottoposte a vari tipi di analisi , la prima delle quali serve per ottenere dal singolo punto (spot), composto da un insieme di pixel, il valore dell'espressione del gene considerato.

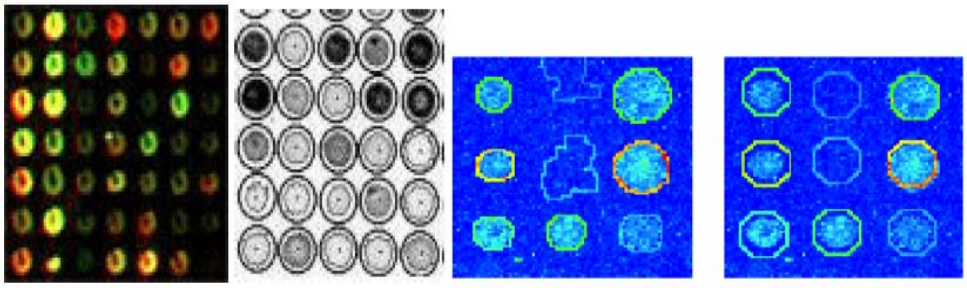


Figura 2.11 – ***Selezione degli spot per la quantificazione del livello di espressione del gene.***

Il punto (spot) viene identificato nella griglia e viene assegnato al gene corrispondente un numero che indica il livello di espressione di quel gene in base ad analisi statistiche dell' immagine dello spot considerato.

Una volta assegnato per ogni colore il livello di espressione del gene, per confrontare i due colori bisogna procedere alla normalizzazione delle due matrici delle intensità.

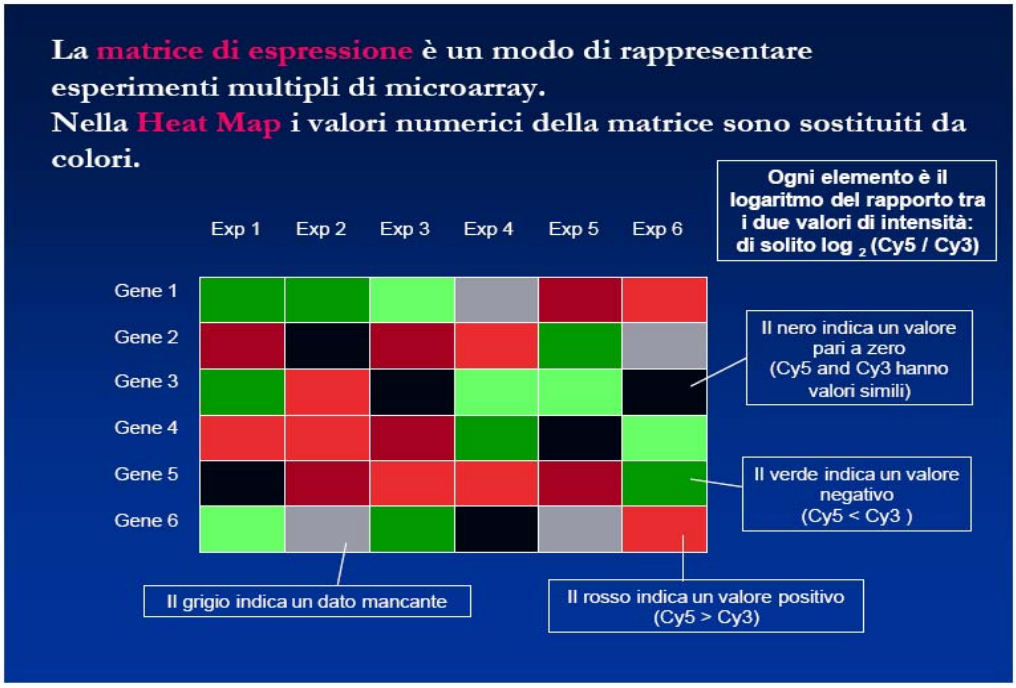


Figura 2.12 – ***Heat Map***

2.2.4 Algoritmi di normalizzazione e di data mining

Con l'insieme dei dati sui geni relativi a molti esperimenti si costruisce la matrice delle espressioni dei geni, in cui le righe rappresentano l'andamento dell'espressione del gene in funzione dell'esperimento e le colonne rappresentano il livello di espressione di ogni gene per il singolo esperimento.

Il valore di un elemento della matrice, espresso con un colore, rappresenta il logaritmo del rapporto del livello di espressione del particolare gene nel particolare esperimento per le cellule da testare e quelle di riferimento.

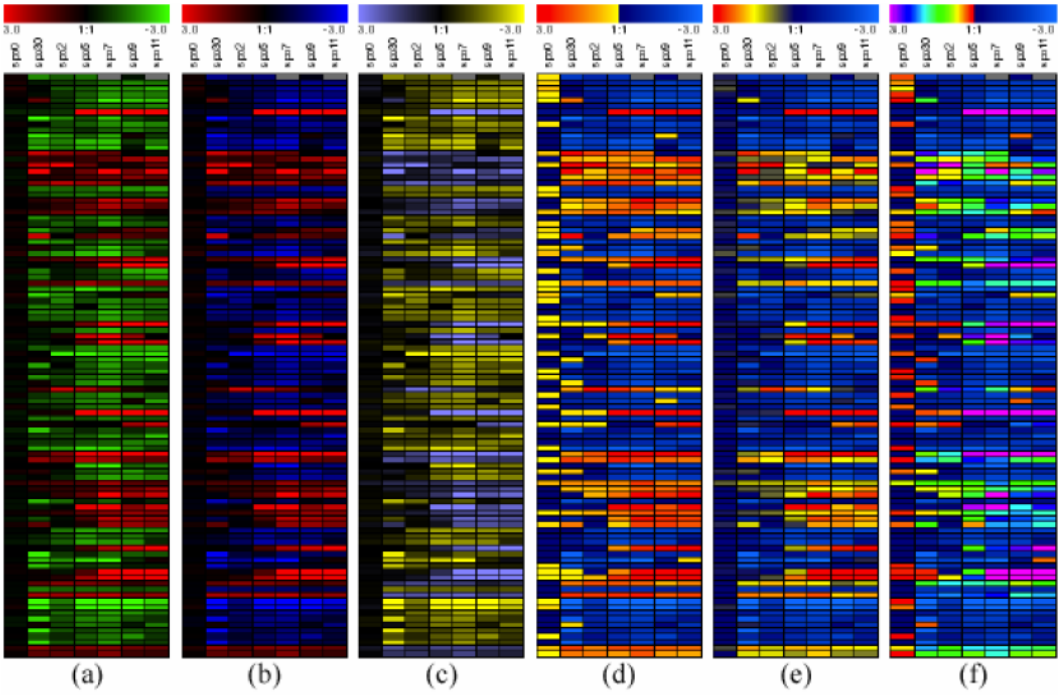


Figura 2.13 – diverse rappresentazioni grafiche per gli stessi dati: ***la più usata (a), per persone daltoniche(b), per risaltare gli estremi(c), per differenziare i geni sotto e sovraespressi (d), evidenzia solo i sovraespressi (e), gradiente solo per i sovraespressi (f).***

Prima dell'analisi statistica dei dati sono spesso usate procedure per la normalizzazione degli stessi.

Le normalizzazioni più usate sono la sottrazione della media dei valori da una espressione o (mean centering) e la divisione dei valori dell'espressione per il valore efficace (o valore RMS) , o per la varianza.

La prima di queste trasformazioni rende confrontabile espressioni di geni che si riferiscono ad esperimenti diversi, permettendo in questo modo di confrontate le variazioni delle espressioni dei geni come è illustrato nella figura seguente.

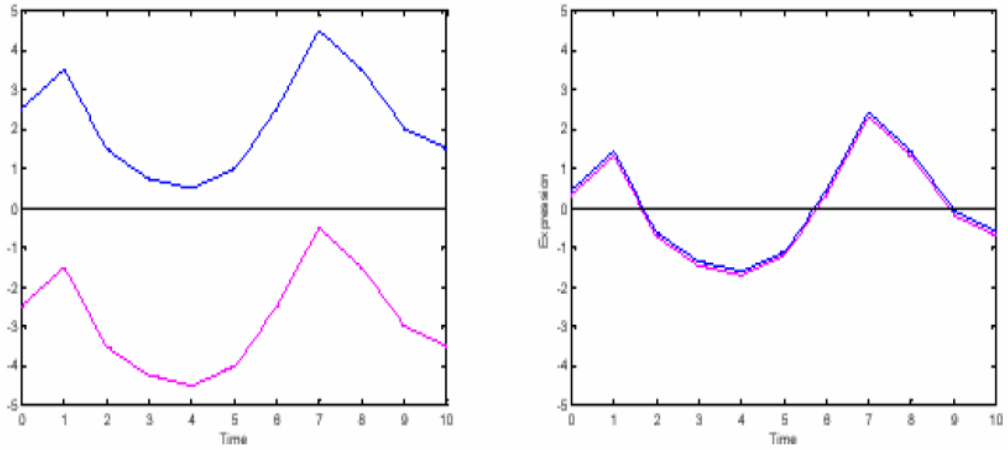


Figura 2.14 – ***(sinistra) espressioni di due geni uno sovraespresso e l'altro sottoespresso, (destra) dopo la normalizzazione la distanza tra loro è nulla.***

Questo tipo di normalizzazione è usato anche per eliminare per un singolo esperimento dominanti di colore dovuti al procedimento tecnologico, non esente da questo tipo di errori.

Un altro tipo di normalizzazione è quello della divisione dei dati per il valore efficace o la per la varianza e questo tipo di trasformazione in una espressione amplifica i valori piccoli e riduce quelli grandi in modulo.

L'inconveniente principale di questo processo è quello di amplificare anche il rumore rendendo le espressioni simili.

Dopo la normalizzazione i dati relativi ad un singolo esperimento sono pronti per essere confrontati con i dati di altri esperimenti ed è questa la vera novità della metodologia.

Si costruisce quindi a partire dai vari esperimenti una matrice che ingloba tutti i dati di molti esperimenti relativi ai molti geni sottoposti ad analisi. Si ha quindi una più complessa analisi dei dati.

Terminata la fase di normalizzazione, che permette il confronto tra array differenti, il passo successivo è trovare i geni differenzialmente espressi tra i campioni. Se per esempio stiamo confrontando una serie di campioni trattati con altri non trattati, cercheremo i geni la cui espressione è affetta dal trattamento, lo stesso se effettuiamo altri tipi di analisi.

Il primo passo per procedere nell'analisi è trovare quali geni mostrino un differente comportamento nelle 2 popolazioni di campioni che stiamo confrontando, filtrare poi i geni immutati e quelli la cui variazione sia nei limiti del rumore di fondo.

La misura più semplice dell'espressione differenziale è il *fold-change* , che rappresenta quante volte il livello di espressione di un gene è maggiore o minore in una classe di campioni rispetto ad un'altra.

Questo semplice confronto non da alcun utilità sull'attendibilità del criterio e così uno smisurato numero di geni che per caso è risultato eccedere la soglia potrebbe essere riconosciuto come differenzialmente espresso.

Per garantire validità alla misurazione è necessario che ogni campione sia replicato, al fine di calcolare la varianza di ogni gene e valutarla con un test statistico.

Fondamentalmente, in un test statistico viene calcolato uno score basato sulle caratteristiche dei dati e quest'ultimo viene confrontato con uno score "atteso" ottenuto dalla distribuzione probabilistica dei dati. Il test è usato per verificare la probabilità di commettere un errore nel ritenere significativo un gene che nella realtà non lo è.

Una fase successiva comprende il clustering dei dati e l'utilizzo di algoritmi di data mining.

Lo scopo di questi ultimi è quello di raggruppare le espressioni dei geni simili. Perché questo?

Perché è qualcosa di molto importante dal punto di vista biologico in quanto indica che o quei geni possono partecipare ad uno stesso processo biologico o che sono regolati dagli stessi meccanismi, il che aiuta a scoprire il funzionamento della cellula e le interazioni tra i geni e le proteine e le interazioni tra la cellula e l'ambiente esterno.

Il concetto di similitudine o di vicinanza è legato alla metrica utilizzata per effettuare la misura della distanza tra le righe della matrice, le espressioni dei geni, che in questo caso vengono interpretate come vettori.

I programmi che implementano questi algoritmi mettono quindi a disposizione anche un gran numero di metriche.

2.3 Descrizione di un esperimento:il MIAME

2.3.1 Premessa

MIAME

Uno dei requisiti fondamentali di una pubblicazione scientifica è che essa sia corredata dalle informazioni necessarie per riprodurre ed analizzare l'esperimento e per individuare i dati che supportano le conclusioni degli autori.

Un esperimento di microarray si compone di dati di espressione genica (TIFF image file provenienti dallo scanning del microarray ibridato, output dell'analisi d'immagine per ogni spot ed ogni canale, matrice di dati di espressione genica combinati) e di metadati, dati riguardanti il design dell'array (descrizione degli spots) ed il design dell' esperimento (campioni, preparazione dell'estratto, labeling, ibridazione).

A questo proposito la società MGED (Microarray Gene Expression Data) ha realizzato un set di linee guide denominato **MIAME** (Minimum Information About Microarray experiment), che specifica le informazioni indispensabili per una corretta e completa annotazione di un esperimento microarray ed un'ontology apposita per l'impiego di una terminologia comune.

Il MIAME (Minimum Information About a Microarray Experiment) è giunto ai giorni d'oggi alla versione 1.1, rilasciata l' 1 Aprile 2002.

Lo scopo di questo documento è sottolineare ed evidenziare l'informazione "minima" richiesta per interpretare in maniera non ambigua, riprodurre e verificare un esperimento di misurazione dell'espressione del gene basato sull'array. Sebbene i dettagli per particolari esperimenti possano essere differenti, il MIAME punta a definire il nucleo di proprietà che è comune alla maggior parte di esperimenti.

Il MIAME non è però una specifica formale, ma una serie di linee guida.

L'obiettivo maggiore del MIAME è guidare lo sviluppo di database di dati prodotti da microarray e il software di gestione degli stessi

2.3.2 La struttura del Miame

Le raccomandazioni fornite dal Miame includono sezioni che normalmente saranno fornite in un formato di tipo testo libero, unite ad informazioni che il Miame consiglia all'utente di dare mediante l'uso massimo di vocaboli controllati o ontologie esterne.

L'uso di un vocabolario "definito" e "controllato" è richiesto per rendere possibili queries sui database e analisi dei dati automatizzate. Poichè sono stati sviluppati completamente solo pochi termini, il MIAME incoraggia gli utenti, se necessario, a fornire le loro proprie definizioni e valori, identificando l'origine della terminologia.

Ciò è ottenuto mediante l'uso di triple del genere
(proprietà, valore, origine)

Un esempio potrebbe essere

??? (proprietà 'tipo di cellula', value: 'epithelial', source: 'Gray's anatomy, 38th ed.'), che è raccomandato rispetto all'aggiunta di descrizioni a testo libero fin quando è possibile.

Ciò permetterà alla comunità di costruire una base di conoscenza dei più utili termini "autorizzati" per la descrizione di esperimenti di microarray.

Comunque il Miame ha 2 sezioni principali:

(1)descrizione dell'array;
(2)descrizione dell'esperimento di espressione dei geni.

Un'altra parte potenzialmente riutilizzabile della descrizione dell'esperimento è quella riguardante i protocolli di laboratorio, includendo i metodi per la processazione dei dati (ad esempio la normalizzazione).

Uno standard per la descrizione dei protocolli, incluso i protocolli per la trasformazione dei dati, è stato sviluppato dalla MGED, per i dettagli http://www.mged.org/normalisation.

La sezione descrizione dell'array descrive:

- design dell'array (nome, tipo di piattaforma, numero di features nell'array, etc.)
- elementi dell'array (spot), distinguendo per essi tre livelli:
- feature –la sua collocazione nell'array,
- reporter –la sequenza nucleotidica presente in un particolare posizione dell'array
- composite sequence –set di reporters usati collettivamente per misurare l'espressione di un particolare gene, esone o una splice variant, riportati attraverso un file ADF (array description file):

La sezione descrizione dell'esperimento di espressione del gene, invece, descrive:

1. disegno dell'esperimento
2. campioni impiegati, preparazione dell'estratto e labeling
3. procedure di ibridazione e parametrizzazione
4. misurazioni di dati e processamento dei dati (raw data, image quantitation table, gene expression data matrix).

Viene raccomandato, per la scelta della terminologia, l'uso di vocabolari controllati ed ontologie esterne. Il Group MGED sta lavorando in tal senso, sviluppando una propria ontologia.

CAPITOLO TRE

IL FRAMEWORK JSF

3.1 Perché la necessità di un Framework?
- 3.1.1 Definizione di framework
- 3.1.2 Introduzione al framework Jsf

3.2 Design patterns
- 3.2.1 Pattern MVC
- 3.2.2 Teamworking

3.3 Il Framework Jsf
3.4 Jsf nel mondo dei frameworks
3.5 Ciclo di vita di una pagina JavaServer Faces
- 3.5.1 Scenari di elaborazione di una richiesta

3.5.2 Ciclo di vita standard
3.6 Expression Language
3.7. Espandibilità del framework
- 3.7.1 Custom Converter
- 3.7.2 Event Listener
- 3.7.3 Custom Validator
- 3.7.4 Custom UI Components

3.8. Sicurezza
3.9 Configurazione dell'applicazione
- 3.9.1 Configurazione dei Backing Beans
- 3.9.2 Localizzazione, Internazionalizzazione e Messaggi
- 3.9.3 Configurare le Navigation Rules

Questo capitolo ha lo scopo di presentare il framework Jsf, ovvero lo strumento utilizzato per la realizzazione dell'applicazione Web, cercando di illustrare le caratteristiche che hanno portato al suo grande utilizzo per la realizzazione di applicazioni Web.

3.1 Perché la necessità di un Framework?

3.1.1 Definizione di framework

Nella produzione del software, il **framework** è una struttura di supporto su cui quest'ultimo può essere organizzato e progettato. Alla base di un framework c'è sempre una serie di librerie di codice utilizzabili con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software, come ad esempio un IDE, un debugger, o altri strumenti ideati per aumentare la velocità di sviluppo del prodotto finito.

Lo scopo di un framework è di risparmiare allo sviluppatore la riscrittura di codice già steso in precedenza per compiti simili, circostanza che si è presentata sempre più spesso man mano che le interfacce utente sono diventate sempre più complesse, o più in generale man mano che è aumentata la quantità di software con funzionalità secondarie simili.

Ad esempio, il tipo di interazione con l'utente offerta da un menu a tendina sarà sempre la stessa indipendentemente dall'applicazione cui il menu appartiene; in casi come questo, un framework che permette di aggiungere la funzionalità di una finestra con un menu a tendina con poche righe di codice sorgente a carico del programmatore, o magari permettendogli di disegnare comodamente il tutto in un ambiente di sviluppo, permetterà al programmatore di concentrarsi sulle vere funzionalità dell'applicazione, senza doversi far carico di scrivere codice "di contorno".

Il termine inglese *framework* quindi può essere tradotto come *intelaiatura* o *struttura*, che è appunto la sua funzione, a sottolineare che al programmatore rimane solo da creare il contenuto vero e proprio dell'applicazione.

In definitiva esistono importanti motivi per l'utilizzo di un framework, che cerchiamo di riassumere in questa maniera:

1. ***Disegno architetturale solido***

Utilizzare un buon framework significa dotare il progetto di fondamenta solide e testate su cui partire.

2. ***Riduzione dei tempi di progetto***

Lo sviluppatore deve implementare solo la logica applicativa, non dovendosi più preoccupare di realizzare componenti infrastrutturali; lo sviluppatore inoltre può tranquillamente ignorare la complessità dell'Object Model implementato nei diversi package integrati nel framework.

3. ***Semplificazione dello sviluppo***

Un buon framework semplifica lo sviluppo applicativo perché fornisce tutta una serie di componenti che risolvono la gran parte dei compiti comuni a tutte le applicazioni web J2EE (controllo del flusso, logging, gestione messaggi di errore, internazionalizzazione, validazione dei dati…); inoltre il framework garantisce l'isolamento, per incapsulamento, di questi componenti proteggendoli da un loro utilizzo improprio da parte dello sviluppatore inesperto.

4. ***Standard nell'implementazione dei servizi applicativi di base***

Utilizzando un framework uno sviluppatore che realizza un servizio applicativo di base deve, generalmente, o estendere classi concrete o astratte del framework, oppure implementare interfacce del framework. Questo consente di ottenere uno standard nell'implementazione dei servizi di base e uniformità nello sviluppo dei servizi anche da parte di sviluppatori diversi.

Affinché un framework sia valido è necessario che abbia determinate caratteristiche:

1. Adozione degli standard

Un framework deve essere fondato sui componenti standard della tecnologia di riferimento e i servizi che mette a disposizione dello sviluppatore devono essere realizzati facendo riferimento ai design-pattern e alle “best pratictises” relative alle attività che devono svolgere.

2. Estensibilità

Un framework, per sua stessa natura, è generico dovendo rappresentare un’architettura generale su cui costruire le proprie applicazioni; è necessario dunque che il framework sia estendibile per poterlo adattare alle proprie esigenze.

3. Documentazione

E’ necessario che un framework sia dettagliatamente documentato, per semplificarne l’utilizzo e renderne comprensibile il funzionamento nel caso il framework debba essere esteso.

3.1.2 Introduzione al framework Jsf

JavaServer Faces (JSF) è un framework per la realizzazione di Web application secondo il pattern MVC (Model – View – Controller).

L'elemento principale, che caratterizza la tecnologia JSF, è l'ampio insieme delle classi e delle relative API (Application Program Interface) che permettono di :

- definire i componenti dell'interfaccia utente (UI) e gestire il mantenimento del relativo stato;
- gestire gli eventi che si verificano sui suddetti componenti, eseguirne la conversione dei valori e la validazione degli stessi;
- specificare la navigazione fra le pagine all'interno della Web application;

supportare l'internazionalizzazione e l'accessibilità;

- realizzare dei componenti personalizzati (Custom Components) che estendano e potenzino le funzionalità delle classi base del framework;

L'architettura complessiva si basa sull'utilizzo delle Servlet :

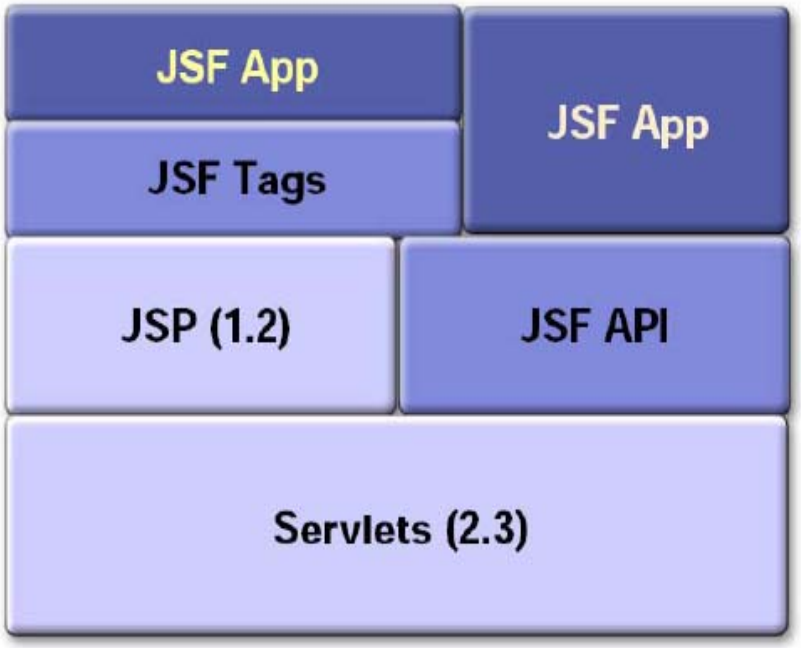


Figura 3.1 – ***Architettura Jsf.***

I vantaggi ottenuti mediante l'utilizzo di JSF sono i seguenti :

- legare i componenti della UI con dati lato server.
- salvare e recuperare lo stato di ciascun componente della UI, attraverso il particolare ciclo di vita di una richiesta JSF (Faces request).
- costruire un'interfaccia utente con componenti riutilizzabili ed estendibili.
- gestire gli eventi che vengono generati dal lato client (client side), con oggetti e codice lato server (server side);

Lo sviluppo di una Web application con JSF, generalmente prevede i seguenti passi fondamentali :

1. Creazione delle pagine JSP utilizzando i componenti della UI ed i tags delle librerie a disposizione.
2. Definire la navigazione fra le pagine dell'applicazione stessa, modificando opportunamente il file di configurazione dell'applicazione "*faces-config.xml*".
3. Sviluppare i backing beans che implementano le funzionalità offerte.
4. Aggiungere le informazioni riguardanti i backing beans all'interno del file di configurazione dell'applicazione.

Inoltre Jsf mette a disposizione due librerie di tag (Tag Libraries):

- Libreria HTML: per l'inserimento dei componenti della UI all'interno di una pagina JSP, secondo il linguaggio HTML;
- Libreria Core: per gestire il legame tra gli elementi dell'interfaccia utente e gli oggetti di back-end dell'applicazione;

3.2 Design patterns

La piattaforma J2EE ha costituito, sin dal suo primo apparire, una vera rivoluzione nell'ambito dello sviluppo di software. Ciò è testimoniato sia dal gran numero di applicazioni che con essa sono state sviluppate, sia dalla continua e costante evoluzione della piattaforma medesima, dato che nel corso del tempo sempre più numerose sono le tecnologie che sono andate a confluirvi.

In particolare l'architetto Christopher Alexander osservò come i suoi colleghi tendevano a risolvere i medesimi problemi, più o meno allo stesso modo. Da tale osservazione, introdusse il concetto di "design pattern", ovviamente riferito all'architettura. Tale intuizione venne applicata allo sviluppo software.

Un pattern è un modello che permette di definire la "soluzione" di un "problema" specifico che si ripresenta, di volta in volta, in un "contesto" diverso.

Presenta inoltre le seguenti caratteristiche :

- il "nome" che individua il pattern e la cui importanza non è secondaria, perché rientra a far parte del vocabolario dello sviluppo software;
- la descrizione del "problema" in maniera dettagliata, con la sua struttura e le condizioni al contorno;
- la "soluzione" che descrive gli artefatti software per risolvere il problema come gli elementi che rientrano nello sviluppo, quali le classi e le relazioni fra esse, le associazioni ed i ruoli, le modalità di collaborazione tra le classi coinvolte ed infine la distribuzione delle responsabilità nella soluzione del particolare problema di design considerato;
- le "conseguenze" che descrivono i risultati che si possono ottenere dall'applicazione del pattern per spingere uno sviluppatore a farne uso;

E' ovvio che esistono numerose tipologie di design patterns, ma nell'ambito dello sviluppo delle Web application, in riferimento all'obiettivo proposto, assume un ruolo rilevante il pattern MVC (Model View Controller).

3.2.1 Pattern MVC

Uno dei principali requisiti di qualsiasi applicazione web è quello di definire un modello applicativo che consenta di disaccoppiare i diversi componenti dell'applicazione in base al loro ruolo nell'architettura, per ottenere vantaggi in termini di riusabilità e manutenibilità.

Esempio tipico di questo problema è l'utilizzo nello sviluppo di una applicazione web J2EE di quei modelli applicativi che nella letteratura sono indicati spesso come "JSP Model 1" e “JSP Model 2”, introdotti con le prime specifiche delle JavaServer Pages.

3.2.1.1 Model 1

Da sole le JavaServer Pages consentono di realizzare applicazioni web dinamiche accedendo a componenti Java contenenti logiche di business o alla base dati del sistema. In questo modello il browser accede direttamente ad una pagina JSP che riceve i dati di input, li processa utilizzando eventualmente oggetti Java, si connette alla base dati effettuando le operazioni necessarie e ritorna al client la pagina html prodotta come risultato della processazione dei dati.

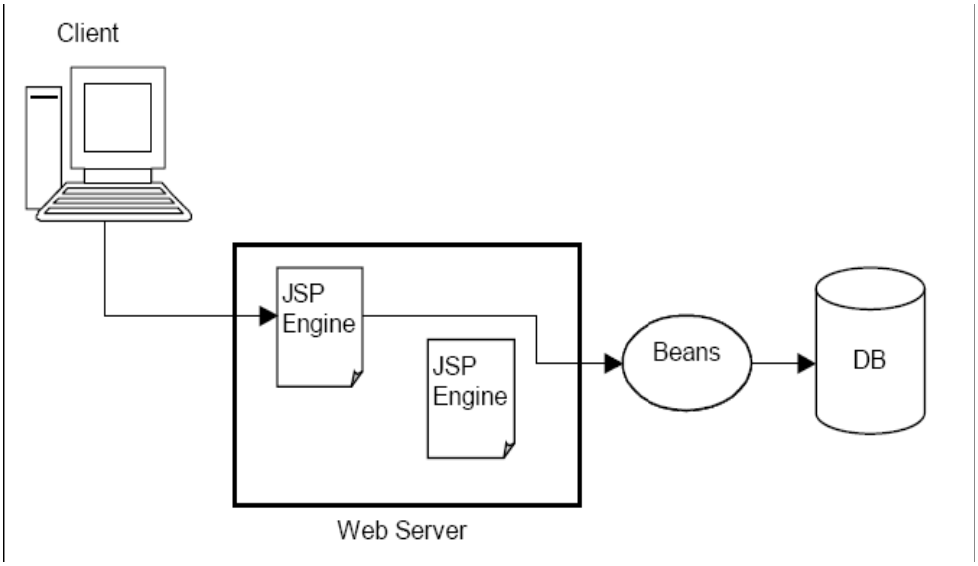


Figura 3.2 – ***Model 1.***

In base a questo modello l'applicazione è costruita secondo una logica "JSP centric" in base alla quale presentation , control e business logic dell'applicazione sono tutti a carico delle pagine JSP.

Il web browser accede direttamente alle pagine JSP dell'applicazione che al loro interno contengono logica applicativa e logica di controllo del flusso.

Quando l'applicazione cresce in complessità non è pensabile svilupparla seguendo un simile approccio.

3.2.1.2 Model 2

Il secondo modello, utilizza le JavaServer Pages come strumento per sviluppare template demandando completamente ad una particolare servlet la processazione dei dati di input.

Nelle architetture Model 2 la richiesta del client viene intercettata prima da una servlet, detta controller servlet, la quale gestisce l'elaborazione iniziale della richiesta e determina quale pagina JSP debba essere visualizzata.

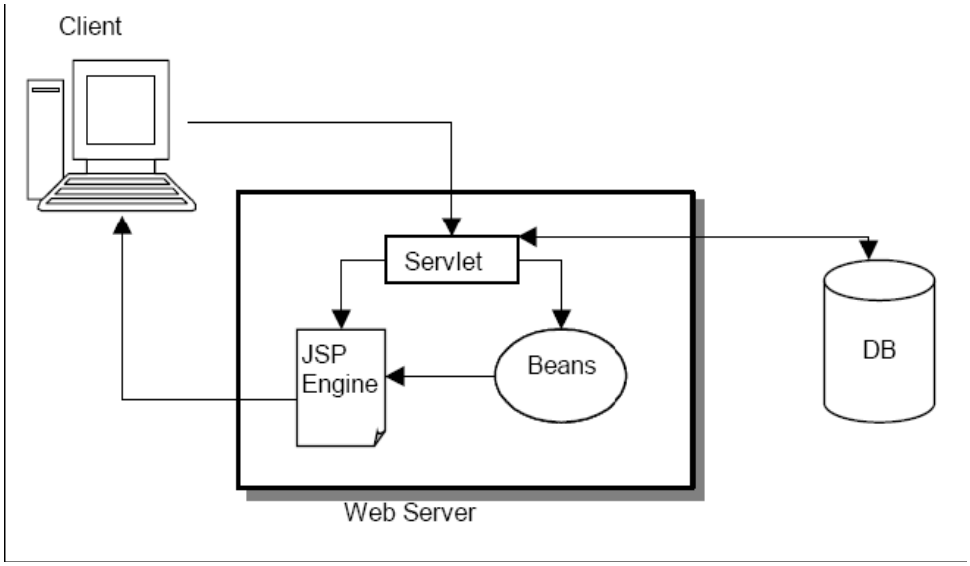


Figura 3.3 – ***Model 2.***

Dunque in questa architettura un client non invia mai direttamente una richiesta a una pagina jsp. Ciò permette alla servlet di effettuare una elaborazione di frontend comprendente autorizzazione ed autenticazione, logging centralizzato e internazionalizzazione.

Una volta terminata l'elaborazione della richiesta, la servlet dirige la richiesta alla giusta pagina JSP. Come si vede la differenza principale tra le due architetture è la presenza di questo punto di accesso ai servizi singolo, rappresentato dalla servlet, il che incoraggia il riutilizzo e la separazione tra i vari strati dell'applicazione.

Proprio per questo il pattern Model 2 è più conosciuto come pattern MVC (Model-View-Controller) Secondo questo pattern, i componenti di un'applicazione vengono logicamente classificati a seconda che siano gestori di logiche di business e di accesso ai dati (model), gestori di visualizzazione dati e dell'input dell'utente (viewer o view) e gestori del flusso di controllo complessivo dell'applicazione (controller).

L'accoppiamento tra i vari livelli è minimo, con evidenti vantaggi in termini di riusabilità, manutenibilità, estensibilità e modularità.

In pratica, si hanno innegabili vantaggi come:

- indipendenza tra i business data (model), la logica di presentazione (view) e quella di controllo (controller);
- separazione dei ruoli e delle relative interfacce;
- viste diverse per il medesimo model;
- semplice supporto per nuove tipologie di client: bisogna scrivere la vista ed il controller appropriati riutilizzando il model esistente.

Di seguito è presentato un diagramma di interazione che evidenza le responsabilità dei tre componenti:

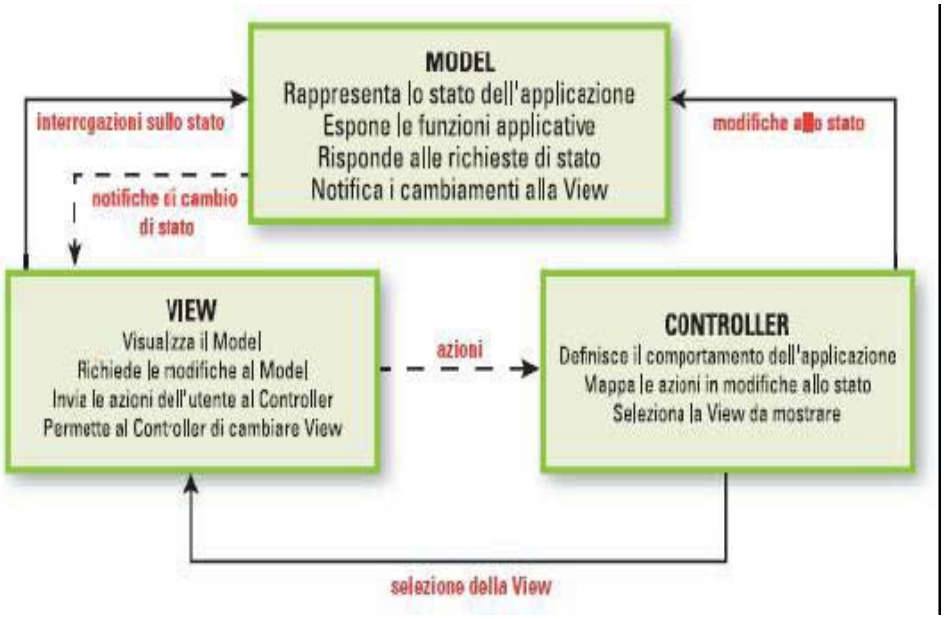


Figura 3.4 – ***Diagramma delle interazioni del modello MVC.***

Precisiamo che tale pattern non è direttamente legato alle web application, anzi, venne introdotto nel mondo del software per la costruzione di interfacce grafiche con Smalltalk-80.

Cerchiamo di analizzare più in dettaglio i 3 componenti fondamentali del pattern MCV:

Model

Analizzando la figura precedente, si evince che il core dell'applicazione viene implementato dal Model che, incapsulando lo stato dell'applicazione, definisce i dati e le operazioni che possono essere eseguite su questi. Quindi definisce le regole di business per l'interazione con i dati, esponendo alla View ed al Controller rispettivamente le funzionalità per l'accesso e l'aggiornamento.

Per lo sviluppo del Model quindi è vivamente consigliato utilizzare le tipiche tecniche di progettazione object oriented al fine di ottenere un componente software che astragga al meglio concetti importati dal mondo reale. Il Model può inoltre avere la responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller, al fine di permettere alle View di presentare agli occhi degli utenti dati sempre aggiornati.

View

La logica di presentazione dei dati viene gestita solo e solamente dalla View. Ciò implica che questa deve fondamentalmente gestire la costruzione dell' interfaccia grafica (GUI), che rappresenta il mezzo mediante il quale gli utenti interagiranno con il sistema.

Ogni GUI può essere costituita da schermate diverse che presentano più modi di interagire con i dati dell'applicazione.

Per far sì che i dati presentati siano sempre aggiornati è possibile adottare due strategie note come "push model" e "pull model". Il push model adotta il pattern Observer, registrando le View come osservatori del Model. Le View possono quindi richiedere gli aggiornamenti al Model in tempo reale grazie alla notifica di quest'ultimo. Benché questa rappresenti la strategia ideale, non è sempre applicabile.

Controller

Questo componente ha la responsabilità di trasformare le interazioni dell'utente con la View in azioni eseguite dal Model. Ma il Controller non rappresenta un semplice "ponte" tra View e Model. Realizzando la mappatura tra input dell'utente e processi eseguiti dal Model e selezionando la schermate della View richieste, il Controller implementa la logica di controllo dell'applicazione.

Determina il modo in cui l'applicazione risponde agli input dell'utente. Esamina le richieste dei client, estrae i parametri della richiesta e li convalida, si interfaccia con lo strato di business logic dell'applicazione. Sceglie la successiva vista da fornire all'utente al termine dell'elaborazione.

Il controller è il “punto di accesso unico” di cui si parlava prima.

3.2.2 Teamworking

Uno dei vantaggi principali che offre la tecnologia JavaServer Faces è, come accennato nel paragrafo precedente, la separazione fra il comportamento (Application processing layer) e la presentazione (Presentation layer –View)di una Web Application, per favorire il teamworking (lavoro di gruppo) ,separando i compiti degli sviluppatori software e dei designers.

Grazie alla divisione strutturale che comporta JSF in una Web Application, lo sviluppo e la manutenzione di quest'ultima possono procedere rapidamente e nel modo più semplice possibile.

Tipicamente i membri coinvolti nella realizzazione di un'applicazione in Jsf assumono i seguenti ruoli:

- *Page author*: colui che usa un linguaggio di markup, come l'Html, ed ha esperienza di sviluppo grafico per la realizzazione delle pagine. Con l'utilizzo di Jsf, egli usa anche i tag delle librerie standard o delle librerie custom, realizzate dagli sviluppatori del software.
- *Application developer*: sviluppa gli oggetti, i gestori degli eventi, i validatori ed i convertitori ed in alcuni casi anche le "helper" class per l'accesso alle basi di dati.
- *Component writer*: ha esperienza con la progettazione dell'interfaccia utente delle applicazioni e preferisce realizzare dei Custom Components per la UI.
- *Application architect:* colui che definisce la struttura dell'applicazione, assicurandone la scalabilità, definendo la navigazione fra le pagine, configurando i JavaBeans e registrando gli oggetti.
- *Tools vendors*: le case produttrici che realizzano gli ambienti di sviluppo che supportano la tecnologia Jsf per poter realizzare l'interfaccia utente in maniera semplice, attraverso il "Drag & Drop" dei componenti.

3.3 Il Framework Jsf

Il framework JSF è costituito da un insieme di classi che sono raggruppate in models :

1. *Execution Model*;
2. *User Interface Component Model*;
3. *Component Rendering Model*;
4. *Conversion Model*;
5. *Event and Listener Model*;
6. *Validation Model*;
7. *Navigation Model*;
8. *Backing Bean Management*

Andiamo ad esaminare brevemente le classi suddette, al fine di comprenderne peculiarità e vantaggi nell'uso.

Execution Model

Questo modello fa riferimento a tutto ciò che riguarda l'esecuzione dell'applicazione, quindi in particolar modo definisce le classi che costituiscono il Controller del pattern MVC.

- *FacesServlet*

Rappresenta la Servlet in esecuzione nel Web Container, che riceve le richieste da parte dei client occupandosi della loro gestione ed assumendo quindi il ruolo di Controller secondo il modello MVC.

Come tutte le Servlet, secondo l'implementazione in Java, essa prevede i metodi di inizializzazione "*init()*", servizio di una richiesta "*service*()" e di distruzione "*destroy*()".

La *FacesServlet* va configurata nel Deployment Descriptor *web.xml* tipico di ogni Web Application realizzata in Java, mediante la seguente definizione :

```
<servlet>
        <servlet-name>FacesServlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
```

Viene specificata la classe e quindi un nome da assegnare alla Servlet, e l'ordine di sequenza di start-up, nel caso in cui la Web Application preveda l'utilizzo di più Servlet per poter gestire richieste di tipo diverso.

Inoltre, la FacesServlet ha il compito di gestire le richieste per file del tipo *.jsf* oppure *.faces* e per questo motivo è necessario specificarne il mapping su di esse :

```
<servlet-mapping>
        <servlet-name>FacesServlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

Con questa configurazione, ogni qualvolta giunga una richiesta di una pagina *.jsf* al Web Container, quest'ultimo passa tale richiesta alla *FacesServlet* che si occuperà di gestirla.

- *Lifecycle - PhaseListener*

Ogni richiesta che arriva alla Web application ha un proprio ciclo di vita (lifecycle) che prevede l'esecuzione di una serie di fasi, fino alla determinazione della risposta. Di tali operazioni, si occupa la classe *Lifecycle,* che riceve ciascuna richiesta dalla *FacesServlet* ed esegue le varie fasi, mediante il proprio metodo *execute().*

Inoltre, ad essa è associata l'interfaccia *PhaseListener*, la quale permette di notificare l'inizio e la fine di ciascuna fase. Tale interfaccia rappresenta un punto di estensione del framework, in quanto è possibile definire alcune elaborazioni specifiche da eseguire prima o dopo una certa fase.

Per fare ciò, è necessario realizzare una classe che implementi l'interfaccia *PhaseListener* ed esegua l'overriding dei metodi *beforePhase()* ed *afterPhase()*, all'interno dei quali sarà contenuto il codice da eseguire.

- *Application*

All'avvio dell'applicazione viene eseguita la lettura, mediante un parsing XML, del file di configurazione e tutte le informazioni in esso contenute vengono caricate all'interno delle proprietà corrispondenti della classe *Application*.

In questo modo, durante l'esecuzione dell'applicazione è possibile accedere a queste informazioni ed eventualmente modificarle.

Le classi principali che sono legate ad essa e che vengono specificate nel file *faces-config.xml* sono le seguenti :

- *ActionListener* : per la gestione degli action events.
- *Lifecycle*, *PhaseListener* : per la gestione del ciclo di vita di ciascuna richiesta.
- *StateManager* : per il salvataggio ed il ripristino dello stato dell'applicazione fra una richiesta ed un'altra.
- *NavigationHandler* : per la gestione della navigazione tra le pagine in base alle *navigation-rules* e agli *outcome*.
- *ViewHandler* : per la gestione delle fasi di ricostruzione della vista di una pagina da uno stato precedente e di visualizzare una risposta.
- *MessageBundle* : che identifica il file *.properties* nel quale ci sono tutti i messaggi da visualizzare nell'applicazione, sulla base di un certo *Locale*, permettendo l'internazionalizzazione.

Se tali classi non vengono specificate nel file *faces-config.xml*, verranno utilizzate quelle di default. In caso contrario, lo sviluppatore ha la possibilità di personalizzare il framework, estendendo queste classi e specificando le proprie classi così ottenute nel file di configurazione.

Inoltre, essa è legata a tutte quelle classi che definiscono gli elementi costitutivi dell'applicazione, in particolare :

- *UIComponent* : che definisce ciascun componente dell'interfaccia utente insieme a tutte le sue classi derivate.
- *Validator* : che implementa un validatore per i dati di input dell'utente.
- *Converter* : che permette di eseguire la conversione dei dati.
- *MethodBinding*, *ValueBinding* : che associano i componenti ed i loro valori, ai metodi o alle proprietà dei backing beans.
- *FacesContext* - *ExternalContext*

La classe *FacesContext* contiene tutte le informazioni che caratterizzano una richiesta e ciò che riguarda la relativa risposta da generare. Essa viene potenzialmente modificata in ciascuna fase del ciclo di vita di una richiesta, per garantire che il contesto sia sempre aggiornato.

Inoltre è legata a tutte le altre classi che si occupano della fase di "rendering" e quindi della produzione di una risposta per una specifica richiesta :

- *UIViewRoot* : rappresenta il nodo radice della view che definisce l'albero dei componenti di una certa pagina;
- *RenderKit* : è una collezione di istanze della classe *Renderer* che si occupano di visualizzare i componenti dell'interfaccia utente, secondo una certa grafica, in relazione al dispositivo client che viene utilizzato per accedere all'applicazione, il linguaggio di markup ed il *Locale* corrente;
- *ResponseWriter* : definisce uno stream attraverso il quale è possibile produrre, direttamente in una pagina di risposta, degli elementi ed i corrispondenti attributi come per i linguaggi HTML ed XML;

- *FacesMessage* : classe che contiene un singolo messaggio associato ad un componente specifico e che va visualizzato in una pagina. Il messaggio in questione può essere tipicamente un messaggio di errore, ad esempio dovuto ad una validazione o conversione errata, ma anche di un qualsiasi altro tipo;

Infine, attraverso il metodo *getExternalContext()*, è possibile ricavare l'istanza della classe *ExternalContext*, la quale definisce il "contesto esterno", ossia tutto ciò che riguarda l'ambiente in cui viene eseguita l'applicazione. Da questa classe è possibile accedere direttamente a tutte le variabili il cui ambito di visibilità (scope) sia quello di tipo *session*, *request* ed *application* e quindi ai parametri ed agli attributi di una richiesta.

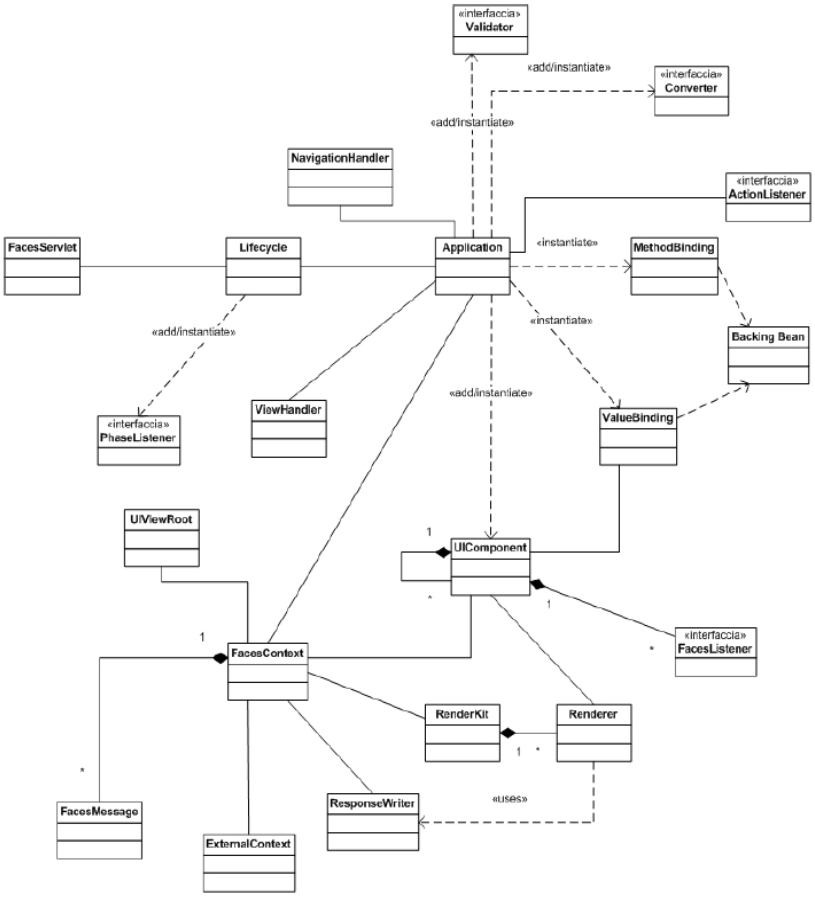


Figura 3.5 – ***Class Diagram Execution Model.***

User Interface Component Model

Il framework JSF permette l'estensione delle classi relative ai componenti dell'interfaccia utente, in modo da potenziare i componenti esistenti aggiungendogli nuove funzionalità.

La classe di base da cui derivano tutte le classi per i componenti standard è *UIComponentBase*, la quale definisce lo stato ed il comportamento di default di un generico componente ed a sua volta estende la classe *UIComponent*.

Di seguito sono riportate tutte le classi che descrivono i componenti previsti in JSF :

UIColumn : generica colonna del componente *UIData*;

UICommand : controllo che può intercettare un action event (es. click de mouse) quando viene attivato;

UIData : permette di gestire un "binding" con una collection di dati per la loro visualizzazione;

UIForm : permette di raggruppare più controlli di una pagina e definire, appunto, un form di invio delle informazioni contenute. E' paragonabile al tag "form" dell' HTML;

UIGraphic : immagine da visualizzare in una pagina;

UIInput : permette di acquisire un'informazione di input dall'utente;

UIMessage : permette la visualizzazione di un messaggio anche sulla base della localizzazione geografica dell'utente (internazionalizzazione);

UIMessages : come il precedente, ma permette di visualizzare un gruppo di più messaggi;

UIOutput : visualizza un'informazione di output in una pagina;

UIPanel : permette di raggruppare più componenti secondo un certo layout;

UIParameter : usato insieme ad altri componenti per specificare una serie di parametri da passare ad essi;

UISelectBoolean : controllo che permette all'utente di selezionare un valore booleano;

UISelectItem : definisce una singola item di una collection di items;

UISelectItems : definisce un insieme di items;

UISelectMany : permette all'utente di selezionare una o più items da una collection di items;

UISelectOne : permette all'utente di selezionare una sola item nell'ambito di una collection di items a disposizione;

UIViewRoot : rappresenta il nodo radice dell'albero dei componenti che, secondo una gerarchia, costituiscono una pagina;

Le classi suddette, oltre ad estendere la classe base *UIComponentBase*, implementano anche una serie di interfacce che ne definiscono il comportamento:

- *ActionSource* : indica che un componente può intercettare un action–event, implementata dalla classe UICommand;
- *ValueHolder* : indica che un componente può immagazzinare un localvalue così come ha la possibilità di accedere al dato corrispondente all'interno del Model, implementata dalla classe UIOutput e da UIInput;
- *EditableValueHolder* : estende *ValueHolder* aggiungendo ulteriori funzionalità ai componenti "editabili", tra cui la validazione e l'intercettazione di eventi del tipo value-change, implementata dalla classe UIInput;
- *NamingContainer* : fa in modo che ciascun componente abbia un identificativo univoco;
- *StateHolder* : segnala che un componente ha uno stato che va salvato ad ogni richiesta, implementata dalle classi UICommand,UIOutput e UIInput;
- *CustomComponent* : per la realizzazione di un componente personalizzato è necessario estendere la classe *UIComponentBase*.

Per poter introdurre un componente all'interno di una pagina, è necessario associare ad esso un tag realizzato mediante una classe nota come Tag Handler.

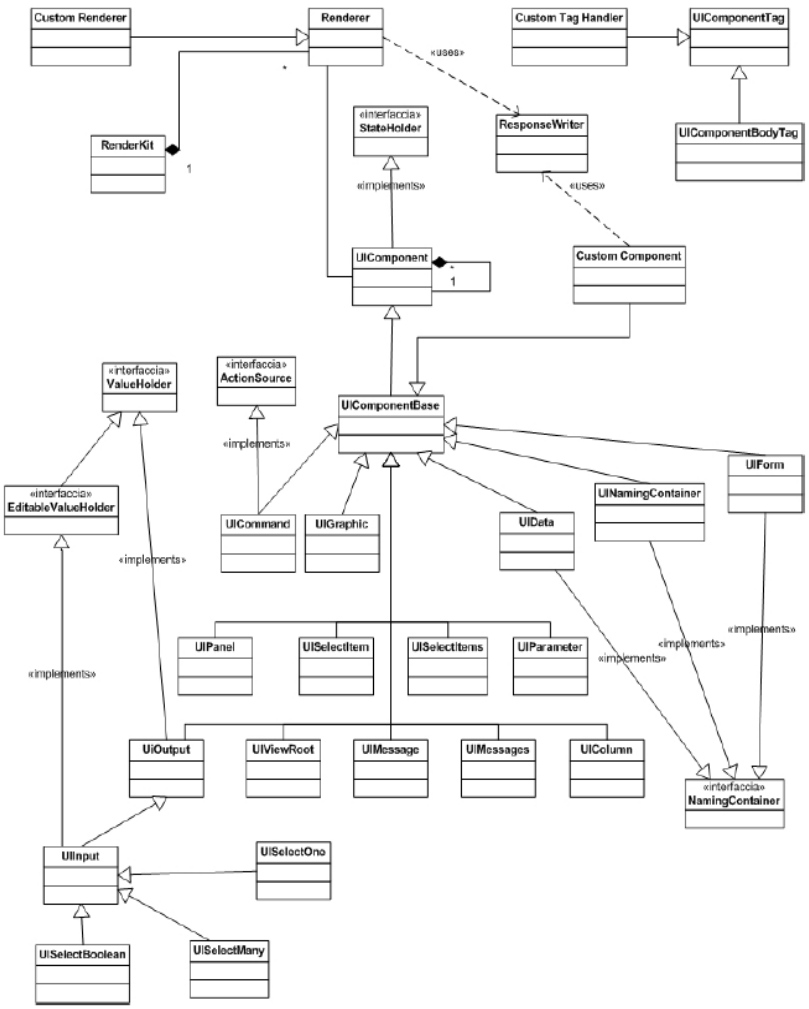


Figura 3.5 – ***Class Diagram UI Component Model /Rendering Model***

Component Rendering Model

La visualizzazione dei componenti viene realizzata mediante delle altre classi specifiche note come render; questo permette di definire una sola volta il comportamento di un componente, ma la possibilità di realizzare diverse visualizzazioni, in relazione ai dispositivi client che possono accedere alla Web application.

La classe che permette di definire un renderer associato ad un componente è la *Renderer*. Essa converte la rappresentazione interna di un oggetto *UIComponent* in una rappresentazione grafica, attraverso uno stream di output sulla pagina.

Mediante la realizzazione di una o più classi che estendono la classe *Renderer*, si possono realizzare diversi renderers che cambiano la modalità di visualizzazione dello stesso componente in base alle necessità o al dispositivo client di accesso.

Un esempio tipico è la classe *UICommand* la cui funzionalità principale è quella di intercettare un action event e permettere un invio di informazioni, così come il passaggio da una pagina all'altra. Questo scopo può essere raggiunto attraverso un "link" oppure un "bottone". Questi ultimi hanno due tag associati per la loro visualizzazione e quindi due classi *Renderer* diverse, che ne permettono una grafica differente, pur basandosi sulla medesima classe componente (appunto *UICommand*).

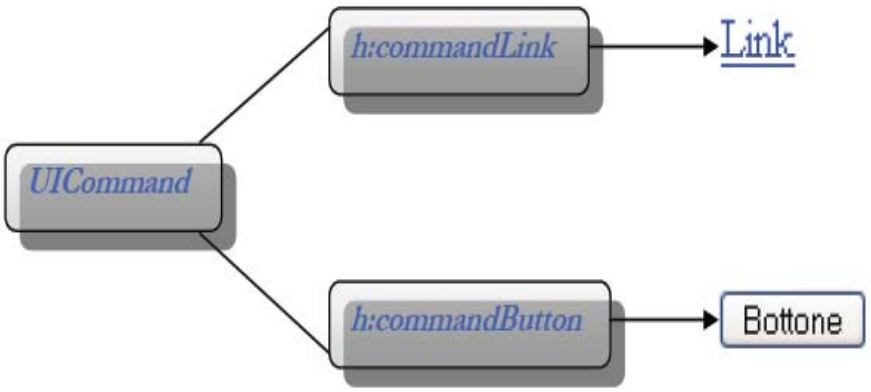


Figura 3.6 – ***Rendering UICommand***

Un insieme di classi *Renderer* costituiscono un Renderer Kit associato ad uno specifico dispositivo di accesso alla Web application, il quale si basa su un determinato linguaggio di markup.

Conversion Model

In una Web application realizzata con JSF ogni componente è associato ad un backing bean, di cui esistono due viste : la model view, che rappresenta un certo tipo di dato all'interno della proprietà del backing bean e la presentation view che rappresenta la modalità secondo la quale il valore può essere letto e modificato dall'utente.

Ad esempio, un componente potrebbe prevedere l'immissione di una data da parte dell'utente. La data sarebbe rappresentata come un oggetto *java.util.Date* all'interno del backing bean (model view) e attraverso una stringa del tipo "yyyymm-dd" nel componente (presentation view).

L'implementazione JSF effettua automaticamente la conversione tra una vista e l'altra; in alcuni casi è possibile che una proprietà di un backing bean non sia di un tipo di dato standard tra quelli previsti dal linguaggio Java, ma sia un di un tipo di dato definito dallo sviluppatore. In queste situazioni, associando il backing bean ad un componente, si rende necessaria la realizzazione dei Custom Converters.

Importanza rilevante in questo contesto è assunta dall'interfaccia Converter, che deve essere implementata da una classe per poter eseguire una conversione Object-to-String (da model view a presentation view) e quella inversa String-to-Object (da presentation view a model view), usando i metodi *getAsObject() e getAsString().*

Ad essa è legata anche la classe *ConverterException* che definisce un'eccezione che può essere sollevata nel momento in cui, per un qualsiasi motivo, la conversione eseguita attraverso i metodi del converter non vada a buon fine.

In tal caso, verrà creato automaticamente un oggetto *FacesMessage*, contenente il messaggio che segnala l'errore, che verrà memorizzato nel *FacesContext* e visualizzato all'utente. Infine, per poter associare un converter ad un componente dell'interfaccia utente, è necessario definire un tag mediante la classe *ConverterTag*.

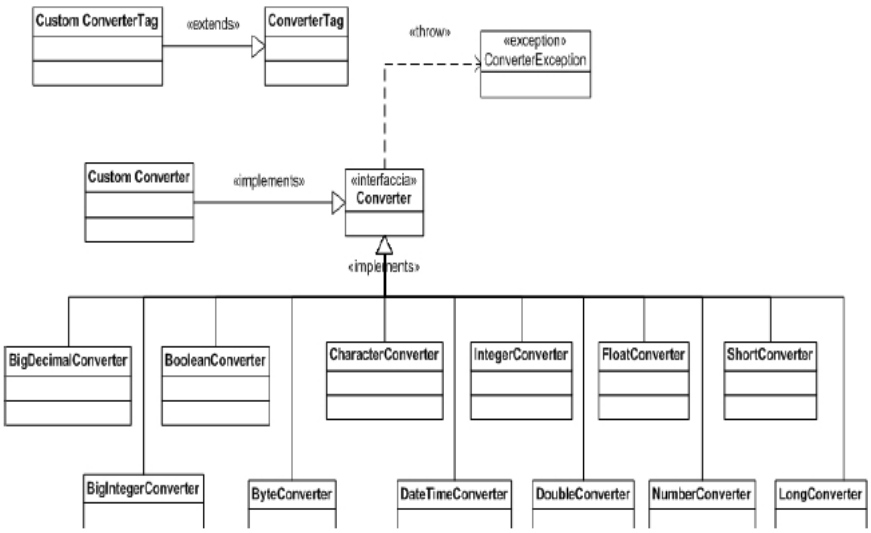


Figura 3.7 – ***Class Diagram Conversion Model***

Event and Listener Model

Questo modello definisce gli eventi (events) ed i relativi gestori (listeners). Nel momento in cui l'utente, interagendo con la UI, fa scatenare l'evento sul componente, entra in gioco il listener che si occupa di gestire l'evento. La tecnologia JSF supporta tre tipi di eventi :

1. *action event* : si scatena quanto l'utente attiva un componente che implementa l'interfaccia *ActionSource*, come nel caso di bottoni o collegamenti ipertestuali (link);
2. *value-change event* : si verifica quando l'utente cambia il valore di un componente implementato dalla classe *UIInput* o da una delle sue sottoclassi, come il caso dei semplici campi di testo, checkbox, radio bottoni, list box, combo box e drop-down list;

3. *data-model event* : si scatena quando c'è uno spostamento ad una nuova riga in un componente della classe *UIData*;

Ci sono due modalità per poter gestire un evento :

1. implementare una classe listener e registrarla sul componente di interesse utilizzando gli attributi *actionListener* oppure *valueChangeListener* del tag del componente stesso;
2. implementare all'interno del backing bean associato al componente, un metodo che debba gestire l'evento;

- *FacesEvent e FacesListener*

E' la classe base che definisce il generico evento che si può verificare su un componente dell'interfaccia utente e memorizza tutte le informazioni che riguardano proprio quest'ultimo.

Da essa derivano due sottoclassi :

1. *ActionEvent* : che definisce un action event;
2. *ValueChangeEvent* : che notifica il verificarsi di un value-change event;

L'interfaccia mediante la quale è possibile realizzare una classe che abbia la capacità di gestire un generico *FacesEvent è la FacesListener*.

Da essa derivano due interfacce :

1. *ActionListener* : interfaccia capace di gestire un action event, relativa alla
classe *ActionEvent*;
2. *ValueChangeListener* : interfaccia per la gestione di un value-change event, relativa alla classe *ValueChangeEvent*;

Le interfacce e le classi listeners in questione sono legate alla classe *AbortProcessingException,* che rappresenta l'eccezione che può essere sollevata nel momento in cui si verifichi un errore durante la gestione di un evento.

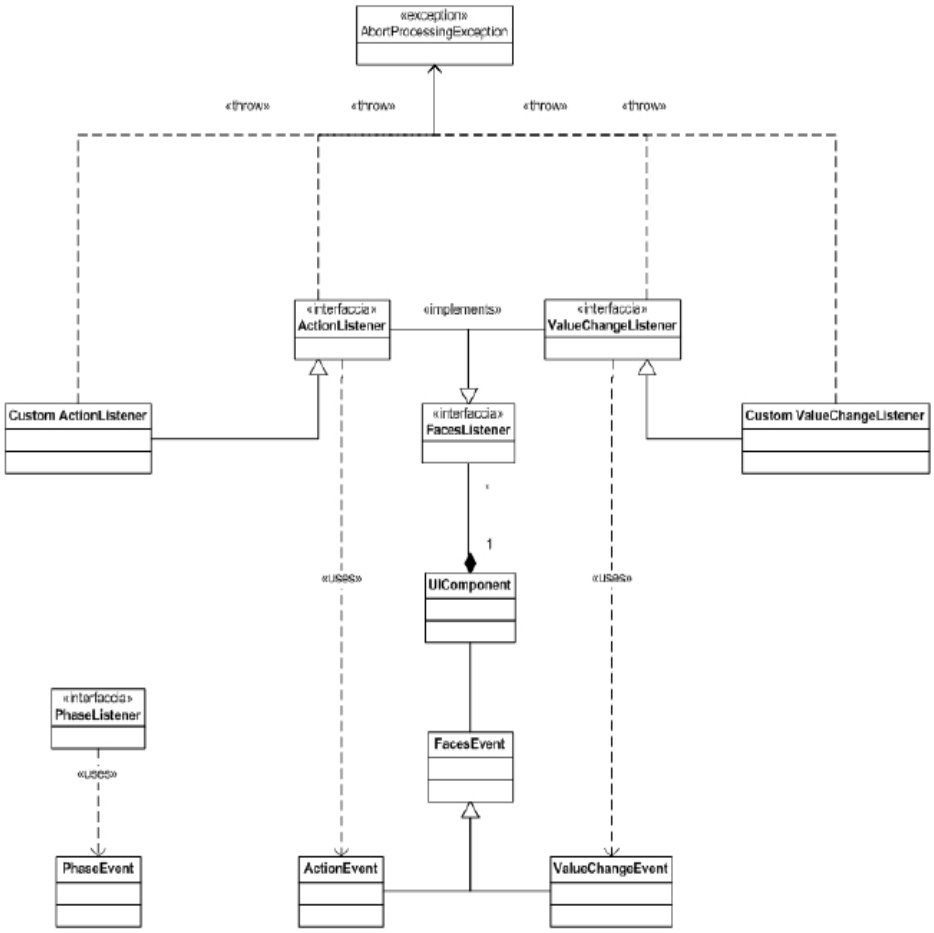


Figura 3.8 – ***Class Diagram Event and Listener Model***

Validation Model

Il meccanismo della validazione viene eseguito prima che il model value corrispondente venga aggiornato ed assuma come valore proprio il local value.

Il framework JSF fornisce una serie di classi standard che possono essere utilizzate per eseguire delle semplici validazioni, in particolare :

- *LengthValidator* : per verificare se il valore di un componente ha una lunghezza che rientra nei limiti specificati;
- *LongRangevalidator*, *DoubleRangeValidator* : per verificare se il valore di tipo long o double di un componente rientra in un range prefissato;

E' comunque possibile realizzare dei validatori personalizzati (Custom Validators) nei seguenti modi :

- realizzare una classe che implementa l'interfaccia *Validator*, registrarla all'interno dell'applicazione e definire un tag che permetta di utilizzare il validatore in una pagina JSP, associandolo ad un componente;
- implementare, all'interno del backing bean associato al componente di cui fare la validazione del valore, un metodo che la esegua;

La prima soluzione risulta quella più interessante, infatti l'interfaccia *Validator* viene implementata da una qualsiasi classe che debba eseguire delle operazioni di validazione, attraverso l'overriding dell'unico metodo *validate()*. Tale metodo può sollevare un'eccezione del tipo *ValidatorException* nel momento in cui si è verificato un problema nel processo di validazione.

Infine, viene utilizzata la classe *ValidatorTag che permette di realizzare* Tag Handler, ossia l'associazione tra validator ed il relativo tag.

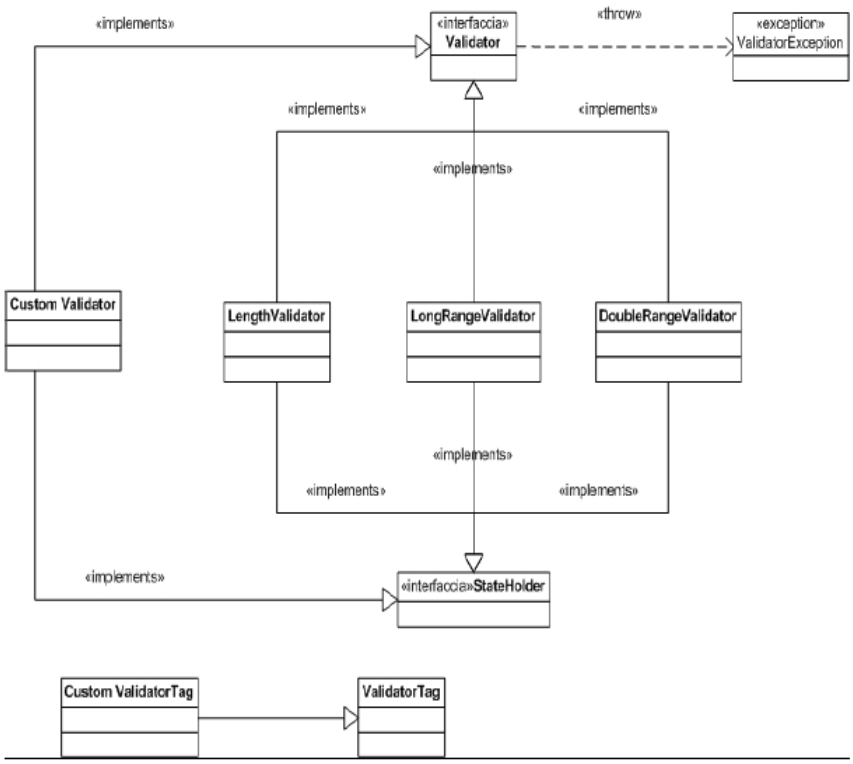


Figura 3.9 – ***Class Diagram Validation Model***

Navigation Model

Questo modello permette di definire la navigazione tra le pagine della Web application mediante la definizione di una serie di regole (*navigation-rules*) e dei cosiddetti "*outcome*".

Le regole specificano come eseguire la navigazione da una certa pagina a un'altra, attraverso le casisistiche di navigazione (*navigation-cases*). Ciascun *navigation-case*, all'interno di una *navigation-rule*, specifica una pagina di destinazione e l'*outcome* che permette di referenziarla per proseguire la navigazione.

Gli *outcome* definiscono una stringa che permette di selezionare una tra le regole ammesse e vengono definiti in corrispondenza di determinati bottoni o link.

Ad esempio, quando si clicca su un bottone oppure su di un link, viene generato un action event gestito dall'istanza *ActionListener* di default che invoca un action-method, implementato all'interno di un backing bean.

Tale metodo esegue eventualmente un'elaborazione e determina l'*outcome* che viene passato al *NavigationHandler* di default, il quale analizza le regole di navigazione, cerca l'*outcome* che gli è stato fornito ed individua la pagina successiva.

Backing Bean Management

Un'applicazione JSF prevede una serie di backing beans che sono associati ad uno o più componenti della UI, in quanto ciascuna delle sue proprietà è legata al valore del componente stesso oppure ad una sua istanza. Inoltre il bean può avere dei metodi che permettono di eseguire ulteriori elaborazioni sui componenti, come la validazione, la gestione degli eventi e il processo di navigazione.

Il binding tra il valore di un componente della UI ,oppure di una sua istanza ,e le proprietà di un backing bean, viene realizzato con la sintassi dell'Expression Language (EL).

Nel primo caso si parla di *value-binding*, mentre nel secondo di *method-binding*.

Se si considera un componente dell'interfaccia utente, immesso in una pagina JSP mediante l'uso di un particolare tag, il suo valore sarà legato ad una proprietà di un backing bean mediante l'attributo *value*, referenziando tale proprietà mediante l'EL. In questo modo, il componente avrà un suo local value ed il corrispondente model value sarà memorizzato nella proprietà del bean.

Nel caso in cui si voglia associare l'istanza del componente e non il suo valore ad una proprietà di un backing bean, si può utilizzare l'attributo *binding*. Inoltre, è possibile utilizzare gli attributi *validator*, *actionListener* e *valueChangeListener* per fare riferimento ad alcuni metodi dei backing bean per poter eseguire delle operazioni di validazione e per poter gestire gli eventi del tipo "action" e "value-change".

Ad esempio, considerando un campo di testo il cui valore deve essere trasferito nella proprietà di un backing bean, dopo essere stato validato attraverso un metodo di quest'ultimo, la sintassi del tag *<h:inputText>* è la seguente :

```
<h:inputText id="idCampoTesto"
value="#{backingBean.proprieta}"
validator="#{backingBean.metodoValidazione}"
valueChangeListener="#{backingBean.metodoEvento}"/>
```

Sulla base di questa definizione, se il valore del componente subisce un cambiamento ed il form che lo contiene viene trasmesso, si scatena un valuechange event che verrà gestito dal metodo del backing bean specificato.

Infine, nel momento in cui si ha a che fare con un componente che implementi l'interfaccia *ActionSource* (es. bottone o link), si può utilizzare l'attributo *action* per specificare il metodo di un backing bean che esegua una certa elaborazione e fornisca come risultato un *outcome* per proseguire la navigazione in una certa direzione.

```
<h:commandButton id="idBottone" action="#{backingBean.metodo}"/>
```

Eseguire il binding fra l'istanza di un componente e la proprietà di un bean permette al bean di modificare gli attributi del componente durante l'esecuzione, ed inoltre gli permette di istanziare un componente.

Invece, eseguire il binding fra il valore del componente e la proprietà del bean permette a colui che ha realizzato la pagina di avere più controllo sugli attributi del componente, i backing beans non sono legati alle classi dei componenti e quindi si ha una netta separazione fra Model e View ed inoltre l'implementazione JSF può realizzare la conversione tra il valore del componente e la relativa proprietà del backing bean in maniera automatica, senza che lo sviluppatore debba realizzare un convertitore, se non in casi particolari.

I backing beans sono configurati nel file *faces-config.xml* che viene valutato all'avvio dell'applicazione, rendendo disponibili tali beans che vengono poi istanziati quando sono referenziati all'interno dei tag. <managed-bean>

<managed-bean-name>
backingBean
</managed-bean-name>
<managed-bean-class>
package.ClasseBackingBean
</managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
</managed-bean>

Gli ambiti di visibilità (scope) in cui sono accessibili i backing beans, sono :

- *request* : un bean è accessibile soltanto in corrispondenza di una richiesta HTTP;
- *session* : un bean è visibile durante un'intera sessione utente;
- *application* : un bean è condiviso ed accessibile fra tutti gli utenti che accedono all'applicazione;

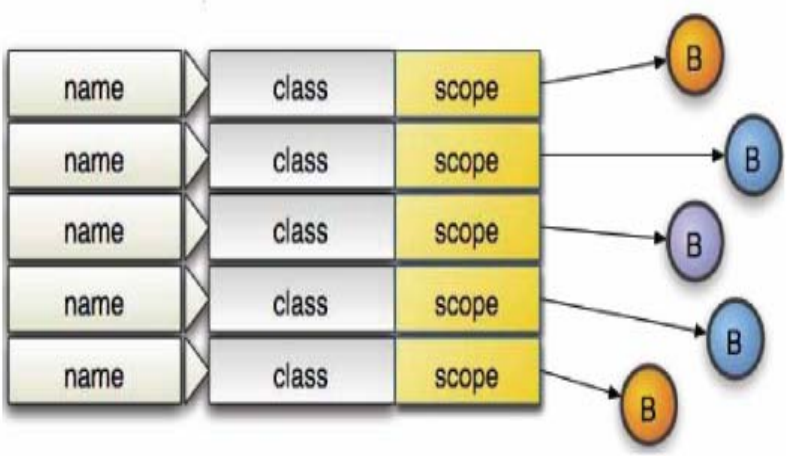


Figura 3.10 – ***Dichiarazione Backing Beans***

La presenza dei backing beans introduce in JSF l'uso dei pattern *Inversion of Control* (IoC), noto anche come *Dependency Injection*.

Tale pattern prevede che, nell'ambito della realizzazione di un software, ci sia sempre una classe nota come "container" che ha il compito di istanziare gli oggetti della business-logic e gestirne la dipendenza e lo scambio di dati fra essi. Nel framework JSF tutto ciò accade in maniera assolutamente automatica.

3.4 Jsf nel mondo dei frameworks

Il mondo dei frameworks per la realizzazione di applicazioni web è molto vasto.

Il nostro ambito di interesse, relativamente a questa tesi, è incentrato sui frameworks che usano java come substrato, o meglio "java oriented".

Quello che sicuramente possiamo dire è che per molti anni Struts è stato il framework più popolare ed ampiamente utilizzato per la realizzazione di Web Application mediante il linguaggio Java.

In tempi recenti è stato introdotto Java Server Faces, che è stato definito in termini di standard ed ha posto la questione di dover scegliere quale delle 2 tecnologie adottare ed in quali condizioni un'applicazione realizzata con Struts dovesse utilizzare le nuove potenzialità di Jsf.

E' impossibile stabilire definitivamente quale dei 2 sia il migliore, perché tale scelta va ponderata tenendo conto del tipo di applicazione da realizzare e della complessità della stessa.

Volendo fare un rapido paragone delle caratteristiche dei due frameworks, potremmo dire che le principali caratteristiche di Struts possono essere così riassunte:

- un'architettura basata sul pattern di sviluppo MVC(Model-View-Controller).
- Capacità di gestione dei form (form management) mediante i cosiddetti formbean che rappresentano lo stato dei dati di input sul server, ai quali va aggiunto un framework per la validazione lato client e lato server.
- Presenza del framework Tiles per la definizione del layout delle pagine mediante dei templates facilmente modificabili, che riducono i tempi di manutenzione grafica dell'applicazione.

- Librerie di tag da utilizzare nelle pagine Jsp per la definizione della View e che lavorano in stretta collaborazione con la parte di form-management e di controllo.
- Ultima, se non principale caratteristica, l'alto elevato di maturità che caratterizza il framework, in quanto è stato realizzato nella sua prima versione nell'anno 2000, diventando uno standard de facto per lo sviluppo di Web application su piattaforma J2EE(Java 2 Enterprise Edition). Un limite può essere dettato dal fatto che Struts rappresenti un'implementazione e quindi sia unico, ma comunque migliorabile ed estendibile.

Dal lato opposto si pone il framework JavaServer Faces, le cui caratteristiche peculiari possono essere così riassunte:

- Architettura basata sul pattern MVC, ma focalizzata in particolar modo alla View.
- Componenti riutilizzabili ed estendibili per la realizzazione dell'interfaccia utente(UI).
- Modello di gestione degli eventi e dei corrispondenti handlers, basato su JavaBeans.
- Utilizzo dell'EL (Expression Language) per realizzare il value-binding e il method-binding, mediante i quali è possibile legare le proprietà dei componenti della UI, rappresentati con tag nelle pagine JSP, direttamente con la business-logic, senza che tali componenti abbiano alcuna conoscenza della struttura delle classi di quest'ultima.
- Ciclo di vita di una richiesta ben definito attraverso una serie di fasi.
- Navigazione delle pagine basata sulla pagina attuale e sulle possibili destinazioni a partire da essa.
- Possibilità di gestione dei backing beans (managed beans) costituenti la business- logic, che possono essere istanziati in un qualsiasi momento "on demand".

La prima release di questo framework è stata definita nel Marzo del 2004, il che evidenzia un livello di maturità ancora basso rispetto a Struts. D'altro canto, JavaServer Faces rappresenta uno standard del quale ne esistono due implementazioni differenti: JSF RI (Reference Implementation) della Sun Microsystems e MyFaces di Apache.

La nostra scelta è caduta sull'implementazione della Sun grazie alla quale abbiamo potuto beneficiare dell'eccellente documentazione fornita al riguardo e del notevole supporto in rete.

Inoltre il cretore di Struts, Craig McClanahan, ha contribuito alla creazione di JSF portando con se un enorme bagaglio di esperienza nella realizzazione di framework per lo sviluppo web.

Nella piattaforma J2EE 5.0 è prevista ora in forma nativa un'implementazione della tecnologia JavaServer Faces.

3.5 Ciclo di vita di una pagina JavaServer Faces

Il ciclo di vita (life-cycle) di una pagina JSF inizia quando il client richiede una pagina, successivamente l'implementazione JSF costruisce la *view*, tenendo conto dello stato degli oggetti relativo ad una richiesta precedente della pagina stessa (viene ricostruito l'ultimo stato della pagina). Il client interagisce con la pagina ed invia i dati; l'implementazione JSF esegue la validazione di questi ultimi e la conversione dei valori in tipi validi dal lato del server. Infine, vengono eseguite le funzionalità della Web application e proposti i risultati al client, nella forma della pagina HTML risultante.

Una pagina JSF è rappresentata da un albero dei componenti che costituiscono l'interfaccia utente, chiamato *view*, caratterizzato da una gerarchia ben definito.

Ad esempio, nella figura che segue, la pagina contenente il form di Login avrà:

- il nodo *root* dell'albero;
- il nodo relativo al form, legato al nodo *root*;
- all'interno del nodo del form, i nodi associati ai tre seguenti componenti :

due campi di testo ed un bottone;

Ovviamente, questa è soltanto una parte della pagina, che potrebbe contenere ulteriori componenti.

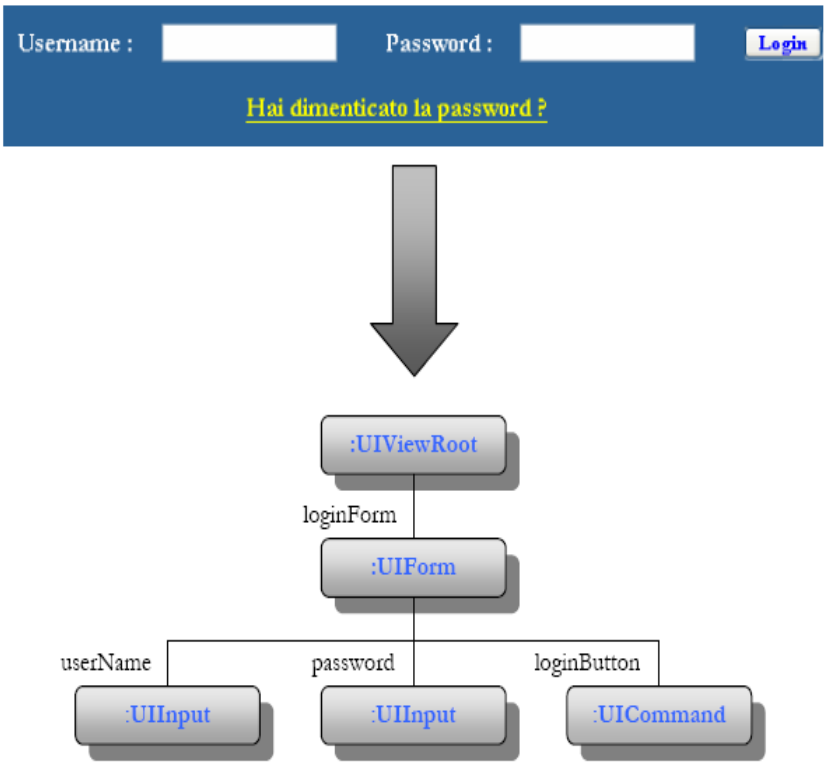


Figura 9 - Esempio struttura della view

Figura 3.10 – ***Esempio struttura della view***

3.5.1 Scenari di elaborazione di una richiesta

Un'applicazione JSF supporta due tipi di richieste (request) e due tipi di risposte(response) :

- *Faces response* : risposta generata durante la fase di *Render Response* del ciclo di vita di una request;
- *Non-Faces response* : risposta che non è creata durante la fase di *Render Response* del ciclo di vita di una request; caso tipico è quello di una pagina JSP che non contiene componenti JSF;
- *Faces request* : una richiesta che viene inviata da una *Faces response* generata in precedenza;
- *Non-Faces request* : richiesta inviata ad un elemento come una Servlet o una pagina JSP e non direttamente ad un componente dell'albero dei componenti JSF di una pagina;

La differenziazione delle richieste e delle risposte suddetta da luogo a tre possibili scenari del ciclo di vita di una richiesta :

Scenario 1 : Non-Faces request genera una Faces response

Un esempio tipico si verifica nel momento in cui si clicca su semplice link HTML per aprire una pagina JSF. In questo caso, entra in gioco il Controller della tecnologia JSF, ossia una Servlet della classe *FacesServlet* che accetta la richiesta e la passa alla classe che implementa il ciclo di vita della stessa per elaborarla.

Quando deve essere generata la *Faces response*, l'applicazione acquisisce i riferimenti agli oggetti necessari alla *view*, memorizza quest'ultima nel *FacesContext* (contesto contenente tutte le informazioni della richiesta) ed invoca il metodo *renderResponse(),* passando alla fase *Render Response*.

Scenario 2 : Faces request genera una Non-Faces response

Talvolta, un'applicazione JSF può avere la necessità di generare una response che non contiene componenti della tecnologia JavaServer Faces. In queste situazioni, lo sviluppatore deve saltare la fase di *Render Response* invocando il metodo *responseComplete()* della classe *FacesContext*. Tale chiamata può avvenire in una delle seguenti fasi : *Apply Request Value, Process Validations, Update Model Values*.

Scenario 3 : Faces request genera una Faces response

Questo è lo scenario tipico, che segue il ciclo di vita standard di una richiesta, in cui un componente JSF invia una request ad un'applicazione JSF utilizzando la *FacesServlet*. Tutti i listeners, validators e converters sono invocati nella fase opportuna del ciclo di vita stesso.

3.5.2 Ciclo di vita Standard

Il ciclo di vita standard di una richiesta è quello dello terzo scenario, che è stato introdotto in precedenza. La tecnologia JSF permette di gestire due tipologie di richieste: richiesta iniziale (*initial request*) e *postback.*

L'initial request è caratterizzata dal fatto che il client richiede per la prima volta una certa pagina e viene gestita eseguendo solo le fasi di *Restore View* e *Render Response*, mentre un postback scaturisce dall'invio di un form che è stato generato da un'initial request precedente e viene gestita eseguendo nell'ordine tutte le fasi.

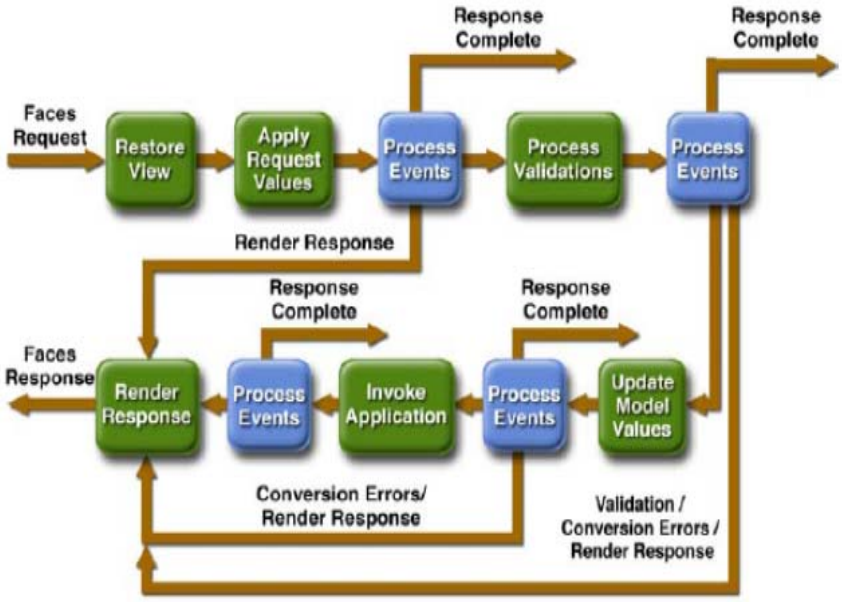


Figura 3.11 – ***Ciclo di vita Standard***

3.5.2.1 Restore View

Rappresenta la prima fase del life-cycle e viene avviata quando una richiesta per una pagina JSF viene generata. Durante questa fase, viene costruita la *view* della pagina (albero degli oggetti associati ai componenti della UI), vengono associati gli events handlers (listener), i validators ed i converters ai componenti ed infine la *view* viene memorizzata nel *FacesContext*, che avrà tutte le informazioni per gestire la richiesta.

Se si tratta di una initial request viene creata una *view* vuota e si passa direttamente alla fase di *Render Response*.

Tale *view* vuota sarà popolata quando, ad esempio, l'utente immetterà dei dati in un form e verrà comandato un postback.

Se si tratta di un postback, la *view* precedente relativa alla pagina già esiste e viene recuperata, utilizzando le informazioni di stato dei componenti della UI che sono salvate sul client oppure sul server (per default).

3.5.2.2 *Apply Request Values*

Dopo aver recuperato l'albero dei componenti (*view*), ogni componente estrae il suo nuovo valore dai parametri della richiesta, usando il metodo *decode()*. Tale valore, noto come *submittedValue*, viene memorizzato localmente al componente stesso. Se la conversione del valore fallisce, viene memorizzato nel *FacesContext* un messaggio di errore che sarà visualizzato nella fase di *Render Response*, insieme ad eventuali altri errori di validazione successivi.

Se un metodo *decode()* oppure un event listener invoca il metodo *renderResponse()* del *FacesContext*, si salta direttamente alla fase di *Render Response*.

Se degli eventi vengono accodati in questa fase, ne verrà eseguito il broadcast verso i corrispondenti listeners, ma la loro gestione avverrà nelle fasi successive.

Se, invece, alcuni componenti della pagina hanno l'attributo *immediate* settato a *true*, allora la validazione, conversione e gestione degli eventi associati vengono processati subito e non posticipati alle fasi successive. A questo punto, se bisogna redirezionare il flusso dell'applicazione verso una risorsa che non ha componenti JSF, viene invocato il metodo *responseComplete()* del *FacesContext*.

Al termine di questa fase, i componenti sono settati ai loro nuovi valori (*submittedValues*) ed i messaggi eventuali di errore e gli eventi sono messi in coda, pronti per essere processati nelle fasi successive.

3.5.2.3 Process Validation

Durante questa fase vengono eseguiti tutti i validators registrati sui componenti della *view*. In primo luogo, vengono valutati gli attributi fissati che specificano le regole di validazione e viene confrontato il valore memorizzato in ciascun componente (local-value) con le regole stesse.

Se il valore locale di un componente non è valido, viene memorizzato un messaggio di errore nel *FacesContext* e si passa alla fase di *Render Response*, per visualizzare tutti i messaggi di questo tipo, insieme ad altri eventuali messaggi dovuti ad errori di conversione nella fase precedente.

Se un metodo *validate()* oppure un event-listener invoca il metodo *renderResponse()* del *FacesContext*, si passa direttamente alla fase di *Render Response*.

A questo punto, se bisogna redirezionare il flusso dell'applicazione verso una risorsa che non ha componenti JSF, viene invocato il metodo *responseComplete()* del *FacesContext*.

Se degli eventi vengono accodati in questa fase, ne verrà eseguito il broadcast verso i corrispondenti listeners, ma la loro gestione avverrà nelle fasi successive.

3.5.2.4 Update Model Values

Una volta che i valori locali dei componenti sono considerati validi, è possibile percorrere l'albero dei componenti stessi ed assegnare alle proprietà dei backing beans tali valori. Ciò vuol dire rendere i *submittedValue* i valori effettivi del modello (model-value). Ovviamente i local-values devono essere convertiti correttamente nei tipi di dati accettati dalle proprietà dei backing beans. Se si verificano degli errori, si passa direttamente alla fase di *Render Response* per visualizzarli, in maniera molto simile agli errori di validazione nelle fasi precedenti.

Se un metodo *updateModels()* oppure un event listener chiama il metodo *renderResponse()* del *FacesContext*, si salta direttamente alla fase di *Render Response*.

A questo punto, se bisogna redirezionare il flusso dell'applicazione verso una risorsa che non ha componenti JSF, viene invocato il metodo *responseComplete()* del *FacesContext*.

Se degli eventi vengono accodati in questa fase, ne verrà eseguito il broadcast (smistamento) verso i corrispondenti listeners, ma la loro gestione avverrà nelle fasi successive.

3.5.2.5 Invoke Application

In questa fase viene gestito ogni evento sia di tipo action che di tipo value-change, come nei casi di invio di un form di click su di un link. Se bisogna redirezionare il flusso dell'applicazione verso una risorsa che non ha componenti JSF, viene invocato il metodo *responseComplete()* del *FacesContext*. Se la *view* è stata recuperata da una richiesta precedente ed su di un componente viene sollevato un evento, questo viene smistato verso il suo listener.

3.5.2.6 Render Response

Durante questa fase, l'implementazione JSF delega il compito della visualizzazione (rendering) della risposta al JSP container, ovviamente nel caso in cui si utilizzino delle pagine JSP.

Se si tratta di una initial request, i componenti della pagina vengono aggiunti alla *view*, mentre nel caso di un postback già sono presenti. In entrambi i casi, i componenti eseguono il proprio rendering sulla base dei tag che li rappresentano nella pagina JSP.

Nel caso di un postback, se nelle fasi precedenti si è verificato qualche errore, viene visualizzata comunque la pagina e se ci sono in essa dei tag *<h:message>* oppure *<h:messages>*, vengono visualizzati i messaggi di errore. Una volta visualizzata la pagina, viene salvato lo stato della response, ossia lo stato dei componenti, sul client oppure sul server, in modo che per un'eventuale successiva request potrà essere eseguita la fase di *Restore View*.

3.6 Expression Language

Il framework JSF utilizza l'Expression Language (EL) mediante il quale è possibile definire delle espressioni per realizzare il value-binding ed il methodbinding tra i componenti dell'interfaccia utente e le proprietà, oppure i metodi, dei backing beans. In queste espressioni viene specificato il nome del bean e la proprietà oppure il metodo a cui fare riferimento, mediante la tipica "dot notation".

#{backingBean.[proprieta | metodo]}

Inoltre, esso fornisce una serie di operatori aritmetici e logici per definire espressioni molto più complesse, magari utilizzando come operandi le proprietà dei backing beans a cui si fa riferimento.

Sono altresì disponibili i seguenti oggetti impliciti (implicit object) ai quali si può fare riferimento da una qualsiasi pagina JSP :

- *applicationScope* : oggetto Map con variabili e beans che si trovano nell'ambito di visibilità di tipo *application*;
- *requestScope* : oggetto Map con variabili e beans che si trovano nell'ambito di visibilità di tipo *request*;
- *sessionScope* : oggetto Map con variabili e beans che si trovano nell'ambito di visibilità di tipo *session*;
- *cookie* : oggetto Map contenente i cookie associati alla richiesta;
- *facesContext* : istanza della classe *FacesContext*;
- *header* : oggetto Map contenente gli header HTTP della richiesta corrente;
- *param* : oggetto Map che ha al suo interno i parametri della richiesta;
- *view* : oggetto che rappresenta il nodo root della *view*;

3.7 Espandibilità del framework

La tecnologia JavaServer Faces offre allo sviluppatore la possibilità di estendere completamente le classi del framework. Nella maggior parte dei casi, per quanto riguarda l'Execution Model, vengono utilizzate sempre le classi di default a meno di particolari esigenze. La potenzialità di personalizzare il framework viene sfruttata soprattutto per quanto riguarda la realizzazione di :

- *Custom Converter*;
- *Event Listeners*;
- *Custom Validator*;
- *Custom UI Components*;

3.7.1 Custom Converter

Il framework JavaServer Faces esegue automaticamente tutte le conversioni necessarie dei valori dei componenti della UI nelle corrispondenti proprietà dei backing beans, in alcuni casi, può nascere l'esigenza di realizzare un convertitore personalizzato, realizzando una classe che implementi l'interfaccia *Converter*.

Tale classe dovrà inoltre eseguire l'overriding dei due metodi seguenti :

- *getAsObject()* : riceve il valore del componente e lo converte nel tipo di dato della proprietà corrispondente del backing bean associato;
- *getAsString()* : riceve il valore della proprietà del backing bean e lo converte in stringa per la visualizzazione nella UI;

```
public Object getAsObject(FacesContext context,
UIComponent component,
String value) {
...
}
```

```
public String getAsString(FacesContext context,
UIComponent component,
Object value) {
...
}
```

Durante la fase di *Apply Request Values*, quando l'utente ha immesso dei valori nei componenti (es. riempiendo un campo di testo), viene invocato il metodo *decode()* del componente oppure del renderer corrispondente, che si occupa di acquisire tale valore. Ovviamente, è in questo metodo che viene invocato *getAsObject()*.

Viceversa, durante la fase di *Render Response*, quando bisogna visualizzare nell'interfaccia utente il valore di un componente, vengono invocati i metodi di encoding del componente stesso oppure del renderer associato, che a loro volta invocano *getAsString()*.

Infine, per rendere disponibile il converter nell'applicazione, bisogna registrarlo nel file *faces-config.xml*. Affinché lo si possa associare ad un componente, è necessario usare l'attributo *converter* del componente stesso oppure il tag *<f:converter>* della Core Library di JSF.

```
<f:converter converterId="idConverter"/>
```

La registrazione di un Custom Converter viene realizzata all'interno del file di configurazione, mediante l'elemento *<converter>*. Quest'ultimo, a sua volta, prevede ulteriori elementi al suo interno, per specificare le seguenti informazioni:

- identificativo univoco del converter per poterlo referenziare nel tag di un componente della UI che ne fa uso (*<converter-id>*);
- la classe che implementa il converter (*<converter-class>*);

<converter>
<converter-id>idConverter</converter-id>
<converter-class>package.ClasseConverter</converter-class>
</converter>

3.7.2 Event Listener

Come visto in precedenza, l'implementazione JSF permette di gestire due tipologie di eventi che si possono verificare sui componenti della UI :

- action event;
- value-change event;

Tale gestione, può essere eseguita in due modi diversi :

- realizzare una classe listener che implementi l'interfaccia *ActionListener* oppure *ValueChangeListener*;
- realizzare, all'interno di un certo backing bean, un metodo che sia capace di gestire l'evento;

3.7.2.1 Implementazione di un ActionListener/ValueChangeListener

Per poter gestire un action event che può essere sollevato da un componente, è possibile sviluppare una classe che implementi l'interfaccia *ActionListener*.

All'interno di essa, sarà eseguito l'overriding del metodo *processAction()*, il quale prevede come parametro di ingresso un oggetto di tipo *ActionEvent* che contiene tutte le informazioni sull'evento e sul componente che l'ha generato.

Ovviamente, all'interno di questo metodo, sarà necessario scrivere il codice per la gestione dell'evento, in quanto il metodo sarà immediatamente invocato, nel momento in cui l'evento sarà intercettato.

```
public void processAction(ActionEvent event)
throws AbortProcessingException {
...
}
```

Una volta realizzata la classe, essa sarà associata ad un componente utilizzando il tag <*f:actionListener*>.

```
<f:actionListener type="classeActionListener"/>
```

Per poter gestire un value-change event per un certo componente, è possibile realizzare una classe che implementa l'interfaccia *ValueChangeListener*.

Tale classe deve semplicemente eseguire l'overriding del metodo *processValueChange()*, il quale ha un parametro di ingresso di tipo *ValueChangeEvent* che contiene le informazioni sull'evento che si è verificato.

Tale metodo viene invocato nel momento in cui l'evento si verifica e quindi conterrà il codice per la sua gestione.

```
public void processValueChange(ValueChangeEvent event)
throws AbortProcessingException {
...
}
```

Infine, per poter registrare la classe listener sul componente, si può utilizzare il tag <*f:valueChangeListener*> della Core Library di JSF.

```
<f:valueChangeListener type="classeValueChangeListener"/>
```

3.7.2.2 Backing Bean Method Listener

Una possibile strada alternativa alla realizzazione delle classi listener, è lo sviluppo di alcuni metodi all'interno dei backing beans che abbiano la capacità di gestire gli eventi.

Nel caso in cui bisogna gestire un action event, è possibile realizzare un metodo che non ha parametri di uscite ed ha come unico parametro di ingresso un oggetto di tipo *ActionEvent* con le informazioni dell'evento intercettato.

Tale metodo potrà essere associato al componente, utilizzando l'attributo *actionListener* del tag che rappresenta il componente stesso all'interno di una pagina.

```
<h:commandLink actionListener="#{backingBean.metodoListener}"/>
```

Se invece abbiamo a che fare con un value-change event, bisogna realizzare un metodo che ha come parametro di ingresso un oggetto del tipo *ValueChangeEvent* e che sarà associato al componente mediante l'attributo *valueChangeListener* del medesimo tag.

```
<h:inputText
valueChangeListener="#{backingBean.metodoListener}"/>
```

3.7.3 Custom Validator

Per eseguire la validazione dei dati, JSF mette a disposizione una serie di validatori standard, ma nel caso in cui questi ultimi non siano sufficienti, è possibile realizzarne di propri.

Le modalità possibili sono le seguenti :

- realizzare una classe che implementi l'interfaccia *Validator*;
- sviluppare un metodo all'interno di un backing bean, che si occupi della validazione;

3.7.3.1 Implementazione di un Validator

Nel caso in cui la scelta sia quella di realizzare una classe che implementa l'interfaccia *Validator*, si deve poi poter associare tale validatore ad un qualsiasi componente che ne possa fare uso. Le possibili soluzioni sono le seguenti :

- se il validator prevede degli attributi e si vuole dare la possibilità al *Page author* di assegnare ad essi dei valori specifici, bisogna realizzare un custom tag mediante il quale è possibile immettere il validatore all'interno della pagina. Tale tag avrà quindi degli attributi accessibili al realizzatore della pagina, secondo una modalità XML-like;

- in alcuni casi si può preferire non realizzare un tag proprietario del validator. In queste situazioni, per associare il validatore ad un componente, si deve utilizzare il tag *<f:validator>* della Core Library , specificando la classe del validatore, ed uno o più tag *<f:attribute>* per specificarne gli attributi;

- un ultimo caso particolare è quello in cui si realizza un Custom Component e gli si vuole associare un validatore integrato.

  Ciò vuol dire che quest'ultimo verrà associato al componente durante l'esecuzione, mediante il metodo *addValidator()*, e non ci sarà la possibilità per il *Page author* di farlo manualmente all'interno di una pagina.

  Questa soluzione "integrata" fornisce un componente potenziato di un validator integrato, ma non da flessibilità al realizzatore della pagina di poter utlizzare il validatore al di fuori del componente.

  Durante la validazione, si potrebbe verificare un errore ed in questo caso sarebbe utile visualizzare un messaggio all'utente. E' possibile creare i messaggi durante l'esecuzione mediante la classe *FacesMessage*, memorizzandoli nel *FacesContext*.

In generale, si preferisce sollevare un'eccezione del tipo *ValidatorException*, nel metodo *validate()* della classe , con il messaggio di errore da visualizzare.

Nel caso più generale possibile, i passi per poter realizzare un validatore sono i seguenti :

1. realizzare la classe che implementa l'interfaccia *Validator*;
2. realizzare un Tag Handler;
3. definire un tag associato al validatore, all'interno di un file TLD (Tag Library Descriptor);

3.7.3.1.1 Class Validator

La classe che implementa l'interfaccia *Validator* deve tipicamente avere una serie di proprietà, che non sono altro che gli attributi del validatore stesso.

Ovviamente, deve essere anche dotata dei metodi di accesso a tali proprietà (Accessor Method), nella forma *get*NomeProprieta*() set*NomeProprieta*()*. Inoltre, bisogna eseguire l'overriding del metodo *validate()* che dovrà contenere il codice per eseguire la validazione.

```
public void validate(FacesContext context,
UIComponent toValidate,
Object value)
throws ValidatorException {
...
}
```

3.7.3.1.2 Tag Handler

Per poter associare un tag al validatore, è necessario realizzare un Tag Handler, che altro non è che una classe che estende la classe di base del framework *ValidatorTag*.

Tale classe deve avere le stesse proprietà della classe validator che saranno gli attributi del tag.

Le caratteristiche principali della classe sono le seguenti :

- i metodi di accesso alle proprietà;
- esegue l'overriding del metodo *createValidator()*, all'interno del quale viene creata un'istanza del validatore e ne vengono inizializzate le proprietà sulla base dei valori che il page author ha assegnato agli attributi del tag;

protected *Validator createValidator()* ***throws*** *JspException {*

...

}

3.7.3.1.3 Tag Library Descriptor

Per poter associare il validatore ad un componente all'interno di una pagina, è necessario realizzare un tag che lo rappresenti.

Fondamentalmente, il tag non fa altro che rappresentare il Tag Handler, il quale avrà poi il compito di utilizzare la classe validator.

Per la descrizione dei tag, vengono utilizzati i file TLD (Tag Library Descriptor) all'interno dei quali, i tag stessi, vengono elencati con i relativi attributi, sfruttando una sintassi XML-like.

<tag>
<name>nomeValidator</name>
<tag-class>package.ClasseValidatorTag</tag-class>
<body-content>JSP</body-content>
<attribute>
<name>attributo</name>
<required>true</required>
<type>package.ClasseTipoAttributo</type>
</attribute>
...
</tag>

3.7.3.2 Backing Bean Method Validator

La validazione del valore di un componente può essere eseguita anche realizzando, all'interno di un backing bean, un metodo che riceve come parametri di ingresso: l'oggetto *FacesContext* dell'applicazione, il componente del cui valore farne la validazione ed infine il valore stesso.

Tale metodo non ha parametri di uscita ed ha la possibilità di generare dei messaggi di errore, nel caso in cui la validazione non andasse a buon fine.

```
public void metodoValidate(FacesContext context,
        UIComponent toValidate,
        Object value) {
...
}
```

Per fare in modo che il metodo venga eseguito, è necessario utilizzare l'attributo *validator* del componente il cui valore deve essere soggetto a validazione.

```
<h:inputText id="idCampoTesto" value="#{backingBean.proprieta}"
validator="#{backingBean.metodoValidate}"/>
```

3.7.3.3 Registrare un Custom Validator

Se l'Application developer ha realizzato dei Custom Validators per eseguire operazioni di validazione, è necessario registrarli all'interno del file di configurazione mediante l'elemento <*validator*>, che conterrà ulteriori tag per specificare le seguenti informazioni :

- identificativo univoco del validator per referenziarlo all'interno della classe che realizza il Tag Handler (<*validator-id*>);
- la classe che implementa il validator (<*validator-class*>);
- gli attributi previsti dal validator, con il relativo nome ed il tipo (<*attribute*>);

Le prime due informazioni sono necessarie, mentre l'ultima è facoltativa considerando che gli attributi del validator sono definiti nella classe che lo implementa e nel suo Tag Handler.

```
<validator>
    <validator-id>idValidator</validator-id>
    <validator-class>package.ClasseValidator</validator-class>
</validator>
```

3.7.4 Custom UI Components

La tecnologia JSF mette a disposizione una serie di componenti standard per la realizzazione dell'interfaccia utente, ma è comunque possibile estendere le funzionalità di questi ultimi o addirittura crearne di nuovi, con delle funzionalità non ancora esistenti.

E' inoltre possibile, creare uno o più renderer per poter visualizzare ciascun componente in maniera diversa su client di tipo differente.

Ciò vuol dire che il comportamento del componente viene definito una sola volta, mentre la sua visualizzazione viene delegata ai renderer che possono essere molteplici.

Ciò non toglie che un componente possa gestire la visualizzazione in maniera autonoma, non facendo uso dei renderer.

In generale, la classe che realizza un componente, ne definisce lo stato ed il comportamento, il quale include : convertire i valori del componente attraverso i convertitori , accodare gli eventi, eseguire la validazione facendo uso dei validatori ed altre funzionalità.

C'è bisogno di creare un Custom Component, in queste situazioni :

1. aggiungere un nuovo comportamento ad un componente standard, ad esempio un nuovo tipo di evento;
2. aggregare più componenti per costituirne uno solo con un unico comportamento;
3. se si ha bisogno di un componente supportato da un client HTML ma non implementato da JSF;
4. realizzare un componente che possa essere utilizzato da un client non HTML, quando magari il componente stesso esiste per un client HTML;

Non c'è la necessità di realizzare un Custom Component in queste situazioni :

5. se bisogna manipolare i dati di un componente o aggiungervi semplici funzionalità. In questo caso, basta creare dei metodi nel backing bean, associato ad esso;
6. bisogna convertire il valore di un componente in un dato non supportato dal suo renderer. In tale situazione, conviene adottare un convertitore standard o realizzarne uno personalizzato;
7. se bisogna eseguire la validazione sul valore del componente, conviene utilizzare un validatore standard o realizzarne uno;
8. se c'è la necessità di registrare degli event listeners su un componente. In tal caso, si adottano le tecniche per la realizzazione degli event listeners;

Quando si crea un Custom Component, bisogna essere sicuri che la classe sia capace di eseguire le seguenti operazioni :

- *Decoding* : prelevare, dai parametri della request, il valore del componente che è stato immesso dall'utente ed associarlo al local value;
- *Encoding* : eseguire il rendering del componente secondo un certo linguaggio di markup, quale può essere HTML , XML , WML in relazione al tipo di client;

JSF supporta due modelli di programmazione per il decoding e l'encoding :

- *Direct implementation* : la stessa classe del componente implementa il decoding e l'encoding;
- *Delegated implementation* : la classe del componente delega l'implementazione del decoding e dell'encoding ad un renderer;

Con il secondo modello, si ha il vantaggio di poter definire il componente una sola volta (stato e comportamento) ma di avere più renderer associati ad esso, che ne permettano la visualizzazione in forme diverse a secondo del device client (browser web , cellulare,…).

Nel caso di un Custom Component, si rende necessaria la realizzazione di un tag che permetta di introdurre il componente in una pagina, operazione opzionale, invece, nel caso in cui si realizzi un validator e quasi mai utilizzata nel caso di un converter. In generale, per poter realizzare un Custom Component, si seguono i seguenti passi :

- sviluppare una classe che rappresenti il componente ed estenda la classe base del framework *UIComponentBase* oppure una delle sue derivate;
- realizzare un Tag Handler;

- eventualmente, sviluppare uno o più renderer estendendo la classe *Renderer*;
- infine, descrivere il tag all'interno di un file TLD;

3.7.4.1 Registrare un Custom Component

Oltre alla registrazione di un Custom Renderer, si rende ovviamente necessaria anche la registrazione del Custom Component a cui esso è associato, mediante l'elemento <*component*>. Per ciascun componente, vengono specificate le seguenti informazioni :

- il tipo del componente (<*component-type*>);
- la classe che implementa il componente (<*component-class*>);
- le proprietà del componente (<*property*>), per ciascuna delle quali è definito il nome (<*property-name*>) e la classe (<*property-class*>);

Tipicamente, le prime due informazioni sono necessarie, mentre le successive sono facoltative, in quanto tali proprietà vengono comunque definite all'interno della classe che implementa il componente.

```
<component>
        <component-type>TipoComponente</component-type>
        <component-class>package.ClasseComponente</component-class>
</component>
```

3.7.4.2 Class Component

Tutti i componenti standard estendono le classi *UIComponentBase* oppure *UIComponent*. Quando si vuole creare un Custom Component, bisogna estendere una di queste ultime se lo si vuole realizzare partendo da zero. E' possibile invece, poter sfruttare le caratteristiche di un componente standard derivando dalla sua classe. Ad esempio se bisogna creare un menù editabile, non conviene estendere la classe *UIComponentBase* ma la classe *UISelectOne*, in modo da sfruttarne il suo comportamento e doverne aggiungere solo le funzionalità necessarie.

Ovviamente, il componente sarà utilizzato all'interno di una pagina ed in generale sarà associato ad un backing bean, che conterrà in una sua proprietà il valore del componente (value-binding) e potrebbe avere dei metodi referenziati dal componente stesso (method-binding).

Il primo passo per la realizzazione della classe del Custom Component è quello di eseguire l'overriding del metodo *getFamily()* che restituisce la famiglia di appartenenza del componente, registrata anche all'interno del file *facesconfig.xml*.

```
public final static String COMPONENT_TYPE ="TipoComponente";
public final static String COMPONENT_FAMILY ="FamigliaComponente";
...
...
public String getFamily() {
        return COMPONENT_FAMILY;
}
...
<component>
        <component-type>TipoComponente</component-type>
        <component-class>package.ClasseComponente</component-class>
</component>
```

Nel caso in cui il componente gestisca in maniera autonoma il rendering, deve occuparsi sia della fase di decoding che di encoding. Per quanto riguarda la fase di deconding, basta eseguire l'overriding del metodo *decode()* che viene invocato nella fase di *Apply Values Change* del ciclo di vita di una richiesta. Per quanto concerne l'encoding, bisogna eseguire l'overriding dei metodi *encodeBegin(), encodeChildre()* ed *encodeEnd()*.

C'è da precisare che, nella maggior parte dei casi, basta realizzare soltanto il primo metodo; i due successivi vengono usati soltanto se il componente ha dei componenti figli al suo interno.

```
...
public void decode(FacesContext context) {
...
}
...
...
public void encodeBegin(FacesContext context) throws IOException {
...
        ResponseWriter writer = context.getResponseWriter();
        writer.startElement("nomeElemento", this);
        writer.writeAttribute("nomeAttributo", valore,"nomeAttributo");
        ...
        writer.write(body);
        writer.endElement("nomeElemento");
        ...
}
```

All'interno di ciascuno di questi metodi, si fa uso di un oggetto *ResponseWriter*, che permette di scrivere direttamente su una pagina, mediante uno stream di output, in un linguaggio XML-like.

Si osservi che il metodo *startElement()* permette di scrivere sullo stream il tag di apertura, usando una o più volte *writeAttribute()* è possibile scrivere gli attributi ed i corrispondenti valori, con il metodo *write()* si scrive il body del tag

ed infine, usando *endElement(),* si scrive il tag di chiusura. In una pagina JSP, il risultato di queste operazioni sarà una cosa del tipo seguente :

```
<libreria:nomeElemento nomeAttributo="[valore]" />
[body]
</libreria:nomeElemento>
```

Se il rendering viene delegato ad un renderer, i metodi di decoding e di encoding non vengono realizzati, ma viene eseguito l'overriding del metodo *getRendererType()* che restituisce l'identificativo del renderer adottato.

3.7.4.3 Tag Handler

La classe Tag Handler associata ad un componente, estende la classe base del framework *UIComponentTag* e guida la fase *Render Response* del ciclo di vita di una richiesta. La prima operazione eseguita è quella di recuperare il tipo di componente associato al tag mediante il metodo *getComponentType().*

```
public String getComponentType() {
        return ClasseComponente.COMPONENT_TYPE;
}
```

Successivamente, mediante il metodo *setProperties(),* vengono inizializzate le proprietà del componente con i valori degli attributi del tag che sono stati specificati nella pagina JSP in cui compare il tag stesso.

```
protected void setProperties(UIComponent component) {
        ...
        componente.setNomeAttributo(this.nomeAttributo);
        ...
        ...
}
```

Infine, tale classe, mediante il metodo *getRendererType()*, restituisce il tipo di renderer adottato, se ne è associato uno al componente, in modo da permettere all'implementazione JSF di eseguire il deconding e l'encoding di quest'ultimo.

```
public String getRendererType() {
        return ClasseRenderer.RENDERER_TYPE;
}
```

Ovviamente gli attributi di un tag possono avere come valore non una stringa, ma un'espressione nel linguaggio EL, per poter eseguire le operazioni di valuebinding o method-bindig e quindi introdurre dei riferimenti alle proprietà di un backing bean oppure ai suoi metodi.

Per ogni attributo che accetti una espressione EL, all'interno del metodo *setProperties()*, per settare l'attributo è necessario utilizzare le classi *MethodBinding* e *ValueBinding*.

L'oggetto *MethodBinding* serve per valutare un'espressione relativa ad un metodo di un backing bean, mentre l'oggetto *ValueBinding* permette di valutare un'espressione relativa ad una delle sue proprietà.

E' raccomandabile che ogni Tag Handler abbia un metodo *release()* per rilasciare le risorse allocate, una volta che siano state utilizzate.

3.7.4.4 Tag Library Descriptor

Per poter utilizzare un componente all'interno di una pagina è necessario realizzare un tag che lo rappresenti.

Fondamentalmente, il tag non fa altro che rappresentare il Tag Handler, il quale avrà poi il compito di utilizzare la classe del componente.

Per la descrizione dei tag vengono utilizzati i file TLD (Tag LIbrary Descriptor) all'interno dei quali i tag stessi vengono elencati con i relativi attributi, sfruttando una sintassi XML-like.

```
<tag>
<name>nomeComponente</name>
<tag-class>package.ClasseComponente</tag-class>
<body-content>JSP</body-content>
<attribute>
<name>attributo</name>
<type>package.classeTipoAttributo</type>
</attribute>
...
</tag>
```

3.7.4.5 Classe Renderer

Nel caso in cui il componente non gestisca il decoding e l'encoding in maniera autonoma, è necessario realizzare un renderer al quale delegare queste operazioni.

La classe che implementa il renderer estende la classe base *Renderer* e deve eseguire l'overriding dei metodi *decode(), encodeBegin(), encodeChildren()* ed *encodeEnd()* per portare a termine il suo compito.

```
public void decode(FacesContext context, UIComponent component) {
        ...
}
...
...
public void encodeBegin(FacesContext context,
        UIComponent component) throws IOException {
        ...
}
```

Rispetto alla signature degli stessi metodi all'interno della classe che realizza il componente, questi ultimi hanno come parametro di ingresso anche l'istanza del componente da renderizzare.

3.7.4.6 Registrare un Custom Renderer in un Renderer Kit

Un Renderer Kit definisce un insieme di classi *Renderer* che vengono utilizzate per poter eseguire la visualizzazione di uno o più componenti della UI, in maniera diversa in base al dispositivo client con cui si accede alla Web application.

Nel caso in cui l'*Application developer* crea un Custom Renderer, per la visualizzazione di un certo componente, è necessario che venga assegnato ad un Renderer Kit, mediante la registrazione nel file di configurazione.

L'elemento XML adottato è <*renderer-kit*>, il quale prevede al suo interno uno o più tag <*renderer*>, ciascuno dei quali contiene le informazioni seguenti relative ad uno specifico renderer :

- la famiglia del componente della UI a cui è associato il renderer (<*component-family*>);
- il tipo del renderer (<*renderer-type*>), che viene utilizzato dal Tag Handler associato al componente, per determinare quale renderer applicare su di esso;
- la classe che implementa il renderer (<*renderer-class*>);
- gli attributi eventuali del renderer (<*attribute*>) con il proprio nome (<*attribute-name*>) e la classe (<*attribute-class*>);

In generale, non è obbligatorio specificare il Renderer Kit di appartenenza di un Custom Renderer; in questo caso, esso sarà assegnato a quello di default HTML, fornito con l'implementazione JSF.

```
<render-kit>
<renderer>
<component-family>FamigliaComponente</component-family>
<renderer-type>TipoRenderer</renderer-type>
<renderer-class>package.ClasseRenderer</renderer-class>
</renderer>
...
...
</render-kit>
```

3.8 Gestione della sicurezza

Per quanto riguarda la sicurezza, un'applicazione realizzata con JavaServerFaces può utilizzare il protocollo HTTPS per il trasferimento delle pagine e delle informazioni, in modo da garantire la crittografia dei dati.

La gestione della sicurezza tramite l'SSL (Secure Socket Layer) è completamente indipendente dal framework ed è praticamente la medesima di una qualsiasi applicazione Web realizzata in Java.

In sostanza, si fa affidamento ai meccanismi del Web Container sottostante, all'interno del quale va abilitato il supporto per le connessioni basate su SSL.

Una volta effettuate questa operazioni, nell'ambito dell'applicazione è necessario modificare opportunamente il Deployment descriptor *web.xml*, per poter specificare quali pagine dovranno utilizzare il protocollo HTTPS.

Per questo scopo, si utilizza il tag <*security-constraint*>, all'interno del quale si specificano le risorse Web da trasmettere tramite SSL (tag <*web-resourcecollection*>) ed il tipo di trasporto da utilizzare (tag <*transport-guarantee*>) , che può essere di tre tipi :

- *NONE* : utilizza HTTP;
- *INTEGRAL* , *CONFIDENTIAL* : utilizza HTTPS;

```
<security-constraint>
    <display-name>SSL Constraint</display-name>
    <web-resource-collection>
    <web-resource-name>
    Automatic SLL Forwarding
    </web-resource-name>
    <url-pattern>[pagina.jsp]</url-pattern>
    <http-method>GET</http-method>
    <http-method>PUT</http-method>
    <http-method>POST</http-method>
```

```
        <http-method>DELETE</http-method>
        </web-resource-collection>
        <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>
```

Questo tipo di approccio evidenzia che la sicurezza è strettamente legata all'architettura sulla quale è in esecuzione l'applicazione e non al framework con cui è stata realizzata.

3.9 Configurazione dell'applicazione

In generale, nell'ambito di una Web application realizzata con JSF, i compiti principali dell' *Application architect* sono tipicamente i seguenti :

- registrare gli oggetti di back-end in modo che siano accessibili da un qualsiasi punto dell'applicazione;
- configurare i backing beans (managed beans) in modo che siano istanziati in maniera corretta quando le pagine fanno riferimento ad essi;
- definire la navigazione tra le pagine dell'applicazione;
- eseguire il "packaging" dell'applicazione, includendo tutte le pagine, gli oggetti e gli altri file, in modo che possa essere distribuita in un qualsiasi web container;

La tecnologia JavaServer Faces mette a disposizione una modalità di configurazione dell'applicazione che è completamente portabile, mediante un documento XML tipicamente chiamato *faces-config.xml*.

In realtà, nel caso di applicazioni di grandi dimensioni, l'applicazione può prevedere anche più di un file di configurazione, ciascuno dei quali permette di configurare una parte delle funzionalità dell'applicazione stessa. E' da ricordare, inoltre, che l'*Application developer* può sempre accedere direttamente da codice al file *faces-config.xml*, grazie all'utilizzo della classe *Application*. Infatti, all'avvio dell'applicazione, un parser XML legge le informazioni del file di configurazione e le carica nell'unica istanza della classe *Application* prevista.

3.9.1 Configurazione dei Backing Beans

Per istanziare i backing beans che vengono utilizzati in un'applicazione JSF ed assegnarli ad un certo ambito di visibilità (scope), è possibile utilizzare la corrispondente funzionalità offerta nel file di configurazione XML. In questo modo, quando una pagina fa riferimento ad un backing bean, l'implementazione JSF lo inizializza in maniera opportuna in accordo alle informazioni contenute nel file *faces-config.xml*.

Attraverso tale funzionalità, si possono ottenere i seguenti vantaggi :

- registrare i beans in un unico file centralizzato, in modo da essere disponibili nell'ambito di tutta l'applicazione o comunque essere istanziati solo nel momento in cui sono necessari;
- personalizzare le proprietà dei beans direttamente nel file di configurazione, senza scrivere del codice aggiuntivo in Java;
- è possibile assegnare dei valori iniziali alle proprietà di un bean, nel momento in cui viene creato;

Per registrare un backing bean nel file di configurazione, si utilizza l'elemento XML *<managed-bean>*, all'interno del quale si utilizzano ulteriori tag per specificare :

- il nome da assegnare al bean, per poterlo referenziare all'interno delle pagine (*<managed-bean-name>*);
- la classe che implementa il bean (*<managed-bean-class>*);
- l'ambito di visiblità (scope) all'interno dell'applicazione (*<managedbean-scope>*);
- le relative proprietà con eventuali valori iniziali (*<managed-property>*);

Le prime tre informazioni sono strettamente necessarie, mentre l'ultima è facoltativa, considerando che le proprietà del bean vengono comunque definite nella corrispondente classe Java. Per quanto riguarda l'ambito di visibilità, esso può essere del tipo : *request*, *session*, *application* e *none* (in questo caso il bean viene istanziato ogni volta che deve essere utilizzato).

```
<managed-bean>
    <managed-bean-name>backingBean</managed-bean-name>
    <managed-bean-class>
    package.ClasseBackingBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

3.9.2 Localizzazione, Internazionalizzazione e Messaggi

Una delle caratteristiche offerte dalla tecnologia JSF è l'Internazionalizzazione (I18N), che permette di visualizzare il testo della Web application in un linguaggio diverso in base alla localizzazione geografica dell'utilizzatore. Per fare ciò è necessario creare uno o più Resource Bundle, ciascuno dei quali contiene i messaggi da visualizzare in un certo linguaggio, tra quelli supportati dall'applicazione.

In generale, i Resource Bundles non sono altro che dei file con estensione "properties", aventi tutti lo stesso nome più il suffisso relativo al linguaggio a cui ciascuno di essi fa riferimento.

Per rendere disponibile questa funzionalità all'applicazione, è necessario registrare i Resource Bundle all'interno del file *faces-config.xml* mediante l'elemento *<message-bundle>*. Ad esso, vanno poi aggiunti il default *Locale* e tutti i restanti *Locale* supportati dall'applicazione, facendo uso di un solo elemento *<default-locale>* ed uno o più

<supported-locale>.

```
<application>
<locale-config>
<default-locale>it</default-locale>
<supported-locale>en</supported-locale>
</locale-config>
<message-bundle>ApplicationResources</message-bundle>
</application>
```

3.9.3 Configurare le Navigation Rules

Come è stato detto nel trattare il Navigation Model, la navigazione all'interno dell'applicazione è definita da un insieme di regole (*navigation-rules*) che permettono di determinare quale sia la pagina successiva, nel momento in cui si clicca su un bottone oppure su di un link. Tali regole sono configurate all'interno del file *faces-config.xml*.

Ogni regola di navigazione specifica in quale direzione muoversi attraverso le pagine, a partire da una certa pagina. Una volta individuata la regola, sulla base della pagina attualmente visualizzata, si determina la pagina successiva sulla base di un *outcome*, ossia una stringa che viene restituita da un metodo di un backing bean, associato al componente (bottone o link) su cui si è cliccato.

Ciascuna regola viene definita all'interno del file di configurazione, mediante l'elemento *<navigation-rule>*, il quale contiene al suo interno ulteriori tag per specificare le seguenti informazioni :

- la pagina dalla quale si parte per la navigazione applicando tale regola (*<from-view-id>*);
- le possibili pagine di destinazione, individuate attraverso dei "casi" di navigazione (*<navigation-case>*), ciascuno dei quali specifica al suo interno : l'*outcome* (*<from-outcome>*) oppure una action (*<fromaction>*) e la pagina stessa di destinazione (*<to-view-id>*);

```
<navigation-rule>
<from-view-id>/pagina.jsp</from-view-id>
<navigation-case>
<from-outcome>nomeOutcome</from-outcome>
<to-view-id>/paginaDestinazione.jsp</to-view-id>
</navigation-case>
...
</navigation-rule>
```

CAPITOLO QUATTRO

PROGETTAZIONE DELL'APPLICAZIONE

4.1 Introduzione alla progettazione di un'applicazione web
4.2 Class diagram
4.3 Use cases Diagrams
4.4 Activity Diagrams
4.5 Sequence Diagrams
4.6 Progetto del database Microarray
- 4.6.1 Costruzione del modello concettuale
- 4.6.2 Dizionario dei dati
- 4.6.3 Vincoli e valori di default
- 4.6.4 Analisi delle ridondanze
- 4.6.5 Costruzione del modello fisico dei dati

Questo capitolo illustra la fase di progettazione dell'applicazione Web realizzata. Questa è la fase più critica di tutto il ciclo di sviluppo software. Una progettazione accurata consente di evitare molti errori e quindi ridurre i tempi e i costi di sviluppo.

La fase di progettazione ha riguardato la realizzazione dei principali diagrammi previsti dalla notazione grafica UML, atti a chiarire gli aspetti e le funzionalità più complesse dell'applicazione. Si è proceduti inoltre alla progettazione dettagliata della base di dati, attraverso tutte le sue fasi e i suoi diagrammi.

4.1 Introduzione alla progettazione di un'applicazione web

L'obiettivo che ci siamo posti in questo lavoro è la progettazione di un'applicazione Web che permetta agli utilizzatori di usufruire delle informazioni relative ad una banca dati di esperimenti genetici basati sul MicroArray, mettendo a disposizione degli utenti e dell'amministratore del sistema una serie di funzionalità atte a favorire la fruizione dei contenuti di interesse.

Per la fase di progettazione ci siamo affidati alle tecniche fornite dall' Ingegneria del software, che può essere così definita:

> *"è l'applicazione di un approccio sistematico, disciplinato, quantificabile allo sviluppo,* alla modellazione e alla messa in *esercizio e manutenzione del software, cioè l'applicazione dell'ingegneria al software. "*

Chiaramente seguendo questa tecnica abbiamo dovuto scegliere tra vari processi di sviluppo che quest'approccio ci metteva a disposizione.

Un processo di sviluppo può essere così definito:

> *"e' una struttura di riferimento più o meno formale e rigida entro la quale si svolgono le attività necessarie a realizzare software di alta qualità e nei tempi stabiliti"*

Nell'ambito del processo possiamo individuare due classi di attività: attività strutturali e attività ombrello.

Le attività strutturali sono delle attività di tipo "mandatory", ossia obbligatorie, dalle quali non è possibile prescindere per lo sviluppo di un sistema software. Queste attività sono tipicamente sequenzializzate, ossia si fissa un ordine cronologico tra queste e i risultati, cioè gli artefatti prodotti da un'attività sono dei dati di input per le attività successive.

Le attività ad ombrello sono quelle attività chiamate così perché possono essere spalmate lungo tutto il processo di sviluppo del sistema, ossia il processo di sviluppo non definisce un momento specifico rispetto al quale svolgere queste attività, ma è durante la pianificazione che si stabilisce quando eventualmente farle.

Queste attività possono essere ripetute più volte durante un ciclo.

Le fasi di sviluppo hanno seguito il tipico ciclo di vita del software secondo il modello "a cascata" o *Waterfall Model* che fa parte della categoria dei modelli prescrittivi.

In realtà abbiamo usato una variante di questo modello, che può essere descritta dalle interazioni in figura

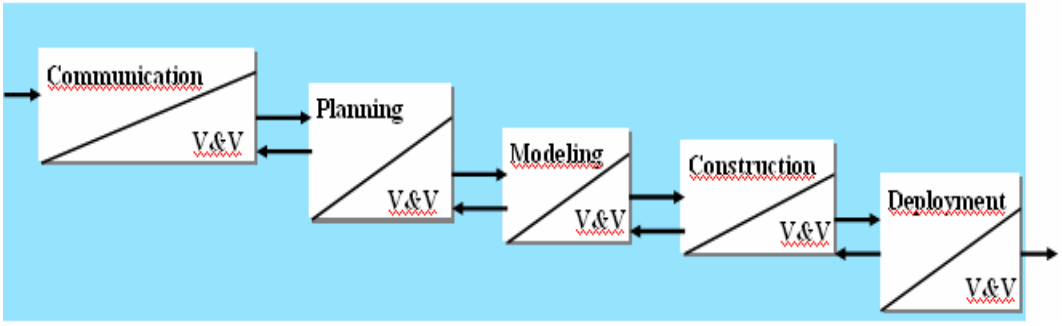


Figura 4.1 – ***Waterfall Model***

Rispetto al modello Waterfall classico ci sono delle varianti, in quanto sono inserite le attività di verifica e di validazione a valle di ogni attività strutturale. Pertanto quando un'attività strutturale si chiude non si passa immediatamente all'attività successiva, ma si va a fare una verifica dei risultati prodotti nell'ambito di quell'attività ed una validazione di quei risultati. Se i risultati vengono validati vengono passati alla fase successiva.

Un'altra modifica rilevante rispetto al modello a cascata rigidamente sequenziale è che da una fase si può ritornare alla fase precedente. Questo significa, ad esempio, che se durante la costruzione ci siamo resi conto di un errore di modellazione possiamo ritornare alla fase precedente, effettuare delle modifiche e ripartire dal ciclo che avevamo interrotto dal punto dove eravamo. Osserviamo che il costo di un errore è tanto maggiore quanto più lontano nel tempo viene rilevato.

Quindi se facciamo un errore nella fase di comunicazione e ce ne accorgiamo solo nella fase di implementazione sarà passato molto tempo tra le due fasi, ossia dovremo ritornare molto all'indietro ed il costo di correzione di quell'errore sarà sicuramente molto elevato.

Se invece ci accorgiamo di un errore di modellazione nella fase di costruzione, siccome queste due fasi sono sostanzialmente vicine il costo di correzione sarà ridotto.

In maniera leggermente più dettagliata le fasi di progetto che si incontrano utilizzando un modello prescrittivo di tipo a cascata possono essere così riassunte:

- studio di fattibilità.
- analisi e specifica dei requisiti.
- progettazione.
- codifica ed implementazione.
- testing.
- messa in esercizio.
- manutenzione;

Dopo aver fatto una panoramica del processo che seguiremo per la realizzazione dell'applicazione MicroArrayWeb scendiamo adesso nei dettagli di ogni singola fase, partendo dalla prima elencata, banale nel nostro caso.

Studio di fattibilità

Lo studio di fattibilità consiste nel cercare di determinare a priori i costi (in tempo e risorse umane) necessari all'organizzazione per portare a termine il progetto.

E' una fase molto importante perché potrebbe consentire di scartare a priori un progetto se non supportato da adeguate risorse per portarlo a termine.

Lo studio di fattibilità, in questo caso, è stato immediatamente superato, in quanto non c'erano da fare considerazioni riguardo le possibili soluzioni da adottare, le risorse finanziarie ed umane a disposizione, tenendo conto che sono stati fissati a priori gli strumenti da utilizzare.

Analisi e specifica dei requisiti

L'analisi e specifica dei requisiti è, in molti processi di sviluppo, un'attività strutturale esplicita.

Lo scopo di questa attività è quello di definire funzioni, vincoli, prestazioni, interfacce e qualsiasi altra caratteristica che il sistema dovrà possedere per soddisfare le necessità del cliente.

A tal fine viene realizzato un documento detto **SRS** (software requirement specification) che formalizza tutti quelli che sono i requisiti software del sistema da realizzare.

Con questo documento si è cercati di individuare tutti i requisiti dell'applicazione, elencando tutte le funzionalità fornite dal sistema software sia all'utente che all'amministratore e considerando per ciascuna di esse una breve descrizione, gli input previsti, le elaborazioni eseguite ed i possibili output prodotti.

4.1.1 Software Requirement Specification (SRS)

Questo documento è fondamentale perché rappresenta il contratto che lo sviluppatore stipula con il cliente.

E' un punto di incontro tra sviluppatore,utilizzatore e cliente che lo discutono e realizzano confrontandosi reciprocamente.

Quando questo SRS viene accettato da tutte le parti in questione, si ha che esso diventa la pietra miliare a partire dalla quale inizia lo sviluppo software del sistema.

L'SRS rappresenta una "garanzia" da parte dello sviluppatore che può, in caso di cambiamenti improvvisi da parte delle esigenze del cliente, attenersi a quanto dichiarato nell'SRS e continuare la strada di sviluppo del progetto intrapresa.

Con un SRS stiamo andando a formalizzare nei minimi dettagli tutto quello che il cliente si aspetta, è una sorta di contratto e va rispettato,onde evitare di incorrere in penali.

Chiaramente questo tipo di documento verrà prodotto quando vogliamo andare ad applicare dei processi di tipo prescrittivo ed in particolare dei processi di tipo a cascata, nei quali nella prima fase andremo a formalizzare tutto quello che deve essere realizzato.

Altra caratteristica fondamentale di questo documento è che fornisce un punto di riferimento per la validazione del prodotto finale.

La cosa fondamentale da sottolineare è che *un SRS di qualità è il pre-requisito per un software di alta qualità.*

Infatti un errore nell'SRS deve produrre errori nel sistema finale. Pertanto un SRS deve essere ben usato oltre che ben scritto e per questo motivo si mette in campo un processo per la realizzazione dell'attività strutturale e quindi per la realizzazione dell'SRS.

Questo processo è detto processo di specifica dei requisiti e si compone fondamentalmente di tre azioni:

l'analisi del problema,
la specifica dei requisiti
la validazione delle specifiche.

Qui di seguito riportiamo **una rappresentazione schematica del processo di specifica dei requisiti:**

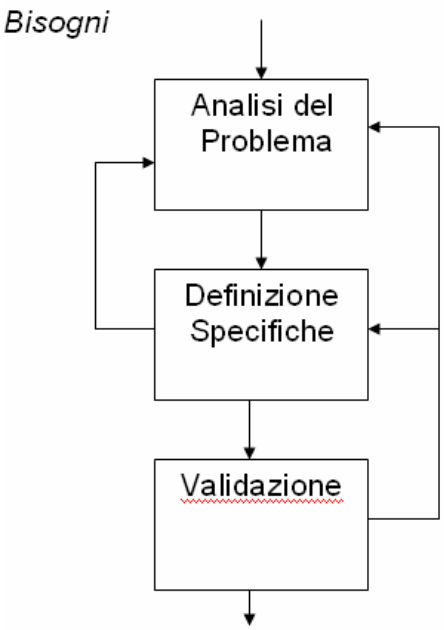


Figura 4.2 – ***Processo di specifica dei requisiti***

Come si può notare dallo schema di cui sopra *il processo di specifica dei requisiti è un processo con feedback***,** quindi è un processo che in ogni fase ci permette di restituire conoscenza alla fase precedente. Questo vuol dire che sicuramente questo processo verrà fatto più volte, inoltre ogni volta che svolgiamo una fase possiamo accorgerci di qualcosa che non va e quindi tornare indietro per rimediare.

QUALITA' DI UN DOCUMENTO SRS

Esistono degli attributi di qualità per l'SRS che sono la correttezza, la completezza, la non ambiguità, la verificabilità, la consistenza, la modificabilità e la tracciabilità. Analizziamoli singolarmente:

- **Correttezza:** un SRS deve essere corretto, questo significa che tutti i requisiti che sono stati specificati all'interno del documento devono essere stati descritti correttamente, ossia devono interpretare esattamente i "desiderata" del cliente e senza ambiguità
- **Completezza:** un SRS deve essere completo perché bisogna raccogliere tutti i desiderata del cliente

- **Non ambiguità:** un SRS non deve essere ambiguo, nel senso che ogni requisito deve avere una sola interpretazione
- **Verificabilità:** un SRS deve essere verificabile, ossia bisogna poter tracciare e verificare che un determinato requisito è stato implementato all'interno del software
- **Consistenza:** un SRS deve essere consistente nel senso che nessun requisito deve essere in contraddizione con gli altri
- **Modificabilità:** un SRS deve essere modificabile, nel senso che ci deve essere la possibilità di operare delle modifiche molto velocemente sui requisiti senza alterare la completezza e la consistenza del documento
- **Tracciabilità:** come già detto in precedenza bisogna poter tenere traccia di un requisito all'interno del software, ossia capire da che pezzo del sistema è stato implementato

Vediamo ora cosa deve sicuramente mettere in luce un SRS, quindi vediamo quali sono le

PARTI FONDAMENTALI DI UN SRS

Un SRS deve mettere in luce una serie di aspetti che sono i requisiti funzionali, i requisiti sulle prestazioni, i vincoli di progettazione e i requisiti sulle interfacce esterne. Anche in questo caso analizziamoli singolarmente:

- **I requisiti funzionali:** sono tutti i requisiti dell'applicazione, classificabili in requisiti normali, requisiti attesi e requisiti interessanti. Dunque la specifica di quali output sono prodotti dal sistema rispetto a dati input e la logica (non l'algoritmo) con cui gli output sono ottenuti. Inoltre sono requisiti funzionali anche il range di input validi e il comportamento rispetto ad input/output non validi.
- **I requisiti sulle prestazioni:** un esempio potrebbe essere il carico che il sistema può sopportare. Si tratta comunque di requisiti che non hanno nulla a che fare con le funzionalità.

Infatti le funzionalità possono essere garantite sia che il sistema fosse capace ad esempio di gestire 1000 richieste contemporanee, sia che il sistema fosse capace di gestire solamente una richiesta. Sono requisiti prestazionali anche quelli derivanti da vincoli temporali, in particolare il sistema potrebbe essere real-time e quindi dovremo avere dei tempi di risposta predefiniti.

- Vincoli di progettazione: sono quelli dovuti ad uno standard da seguire, oppure a dei limiti nelle risorse hardware/software impiegate, oppure ancora dovuti all'ambiente operativo, all'affidabilità e alla sicurezza.

- Requisiti sulle interfacce esterne: quando si costruisce il sistema e si specificano quali sono le funzionalità del sistema, bisogna anche indicare chi o cosa deve essere in grado di sollecitarlo con degli eventi esterni.

Ad esempio possiamo volere che il sistema sia utilizzato soltanto da esseri umani, allora in tal caso dobbiamo dotare il nostro sistema di interfacce grafiche; se invece ad interagire con il sistema sono altri sistemi, ad esempio hardware, allora avremo dei segnali hardware, ossia delle interfacce hardware, attraverso le quali si avrà interazione e colloquio. In definitiva, definite le funzionalità del sistema, ci sono una serie di possibili interazioni diverse con questo sistema. La formalizzazione delle interfacce è dunque un altro aspetto rilevante del documento SRS.

I vari punti di questo documento però non sono tutti obbligatori. Ad esempio se vogliamo realizzare un sistema che lavori a run-time e che quindi non abbia problemi di persistenza dei dati, allora vuol dire che non ci sarà bisogno di un database e quindi nell'SRS potremo eliminare la voce inerente al database.

Dunque la struttura del documento di cui sopra va presa come riferimento, però ovviamente vanno riempiti soltanto i paragrafi che hanno un senso per il nostro progetto.

Nell'appendice a) sarà presentato l'SRS realizzato per la nostra applicazione, riferendoci allo standard Std IEEE-SRS-1998.

Progettazione

La fase di progettazione ha previsto la realizzazione dei principali diagrammi UML, in relazione soprattutto all'analisi del software e non tanto alla sua corrispondente implementazione. Essi costituiscono un'astrazione di alto livello del sistema e sono completamente indipendenti da quelli che possono essere gli strumenti utilizzati nella codifica.

In particolare abbiamo trovato adeguato per la nostra applicazione soffermarci su:

- Class Diagram;
- Use Case Diagram;
- Activity Diagram;
- Sequence Diagram;

che per l'applicazione esaminata ci sono sembrati più che sufficienti ed esaustivi per chiarire i punti più "ambigui" o "complessi" del programma.

codifica ed implementazione

Lo sviluppo ha poi previsto la fase di codifica, realizzando l'applicazione con il framework JSF ed il linguaggio Java.

testing

La fase di testing ha permesso la correzione di errori e problemi rilevati durante il funzionamento del software.

messa in esercizio e manutenzione

Infine sono state previste delle vere e proprie fasi di messa in esercizio e manutenzione presso il Cnr di Avellino, dove l'applicazione è stata testata, revisionata e, dove necessario, migliorata per andare incontro alle esigenze dei tester.

Analizziamo ora più in dettaglio la parte di progettazione, con la realizzazione dei relativi diagrammi.

4.2 Class diagram.

Mediante il Class Diagram è possibile definire tutte quelle che sono le entità caratteristiche del sistema software e le relazioni che ci sono tra di esse.

Ciascuna entità è ovviamente modellabile attraverso il concetto di classe, che ne definisce le caratteristiche (i dati) ed il comportamento (le operazioni).

Ovviamente, anche il Class Diagram può essere realizzato a diversi livelli di astrazione, passando da un livello elevato ad un livello più basso e strettamente legato alla fase di implementazione e quindi relazionato al framework ed al linguaggio utilizzato, come quello che abbiamo deciso di presentare.

Attraverso questo diagramma è possibile modellare la vista statica del sistema, tenendo anche conto di quelle che sono le funzionalità offerte da quest'ultimo e rese disponibili attraverso le entità che lo compongono.

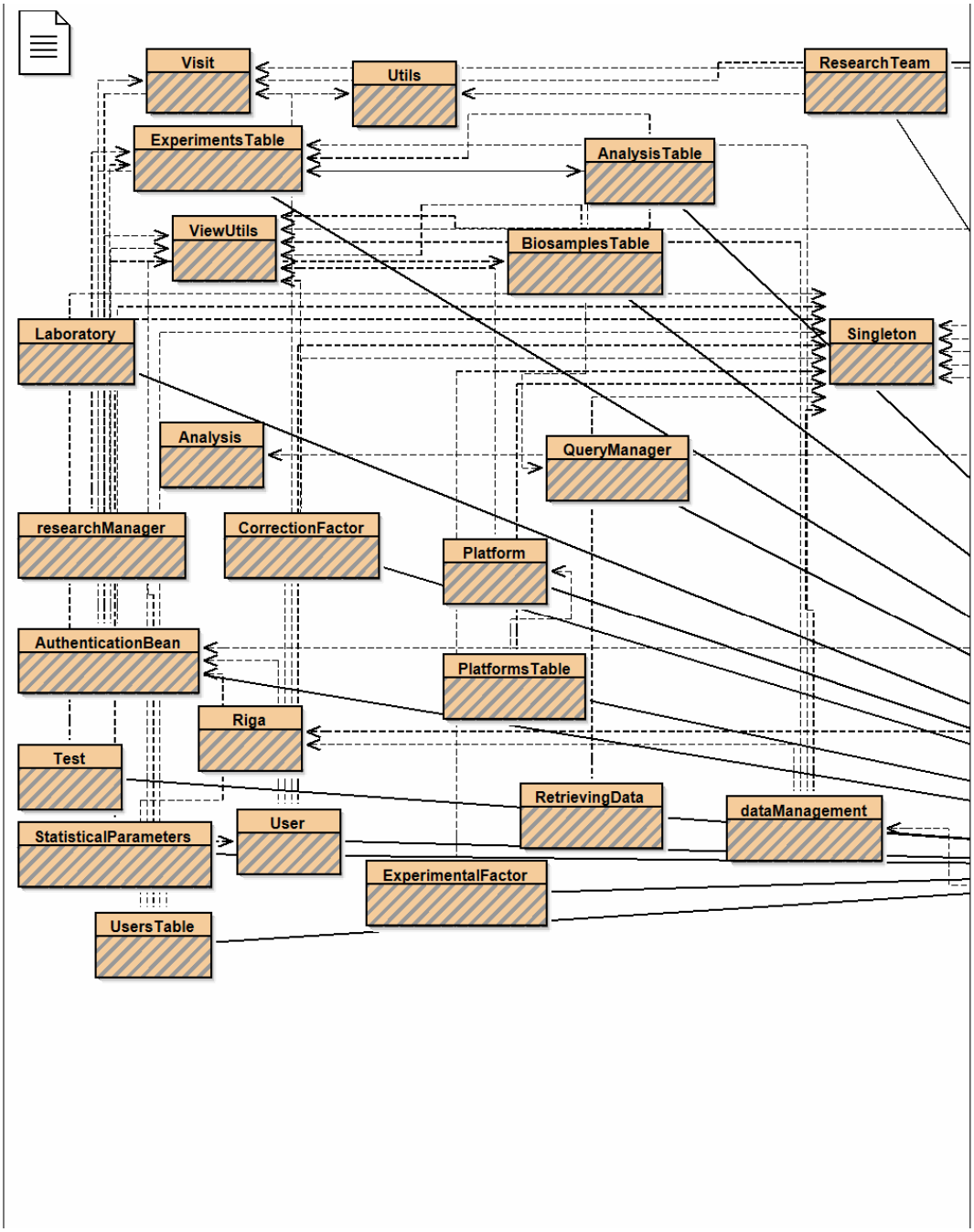


Figura 4.3a – ***Class Diagram dell'applicazione***

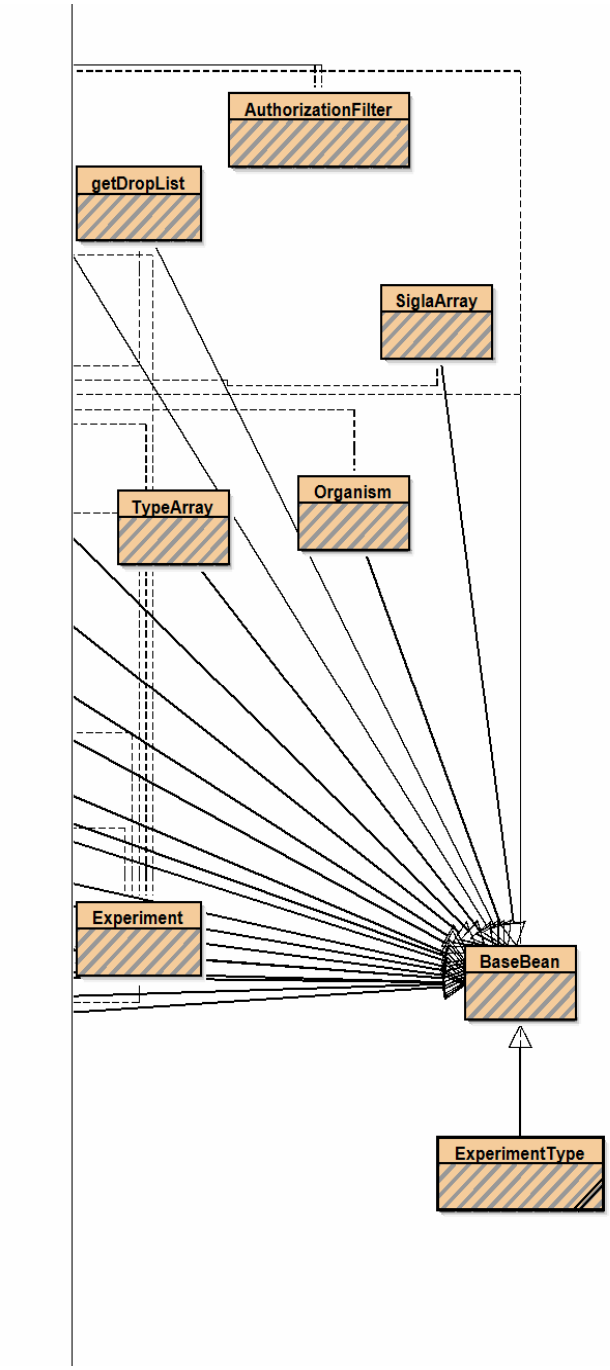


Figura 4.3b – ***Class Diagram dell'applicazione***

4.3 Use Cases diagram.

Partiamo da una semplice definizione di caso d'uso:

"i casi d'uso sono una tecnica utilizzata per identificare i requisiti funzionali di un sistema e si basano sulla descrizione, in una forma quasi narrativa, delle interazioni tipiche tra gli utenti e il sistema".

Fondamentale per la comprensione dell'essenza di un caso d'uso è la formulazione della definizione di scenario:

"uno scenario è una sequenza di passi che caratterizzano una particolare interazione tra l'utente e il sistema".

Una semplice operazione del sistema può portare delle complicazioni durante la sua esecuzione che danno luogo a diversi e molteplici scenari.

Un caso d'uso è un insieme di scenari che hanno in comune lo scopo finale dell'utente.

I casi d'uso rappresentano una visione del sistema dall'esterno, per cui non vi sarà alcuna corrispondenza tra i casi d'uso e le classi che comporranno il sistema.

I diagrammi dei casi d'uso (Use Case Diagrams) costituiscono uno strumento utile per catturare il comportamento esterno del sistema da sviluppare, senza dover specificare come tale comportamento debba essere realizzato; il sistema è visto come una scatola nera (black-box).

Essi forniscono una descrizione dei "modi" in cui il sistema potrà essere utilizzato, ossia ciò che l'utente potrà fare su di esso e come quest'ultimo risponderà alle sollecitazioni.

La realizzazione dei diagrammi ha previsto un approccio top-down, partendo dallo scenario di carattere generale, in cui le modalità d'uso logicamente correlate sono accorpate, fino ad una decomposizione in cui sono descritti i casi di utilizzo non ulteriormente scomponibili.

Gli use cases possono essere rappresentati in due modi, con una modalità testuale e/o con una modalità grafica.

Nella realizzazione di prodotti software, e quindi anche nella applicazione oggetto della tesi, è buona norma utilizzare entrambe le rappresentazioni.

Il primo passo per la costruzione dei Casi d'Uso è rappresentato dall'individuazione degli attori, ovvero degli utilizzatori del sistema.

Gli attori che utilizzeranno l'applicazione sono classificabili in due tipi:

l'Amministratore del Sistema
l'Utente generico

Figura 4.4 – ***Ruoli nell'applicazione***

In realtà l'Amministratore del Sistema rappresenta una specializzazione dell'Utente generico, come si può notare dalla figura 4.1, in quanto è un Utente generico con delle funzionalità aggiuntive.

Per entrambi gli utenti non sono richieste particolari conoscenze informatiche, tranne la familiarità con gli ambienti a finestre e la capacità di utilizzo di un browser Web.

Partiamo quindi da un caso d'uso generale che riassuma tutte le funzionalità del sistema per scendere, dove ritenuto necessario per la comprensione, a livelli di dettaglio maggiore, focalizzandoci sui componenti dei casi d'uso più complessi.

4.3.1 Caso d'uso Generale

Facendo riferimento ai requisiti assegnati, è stato definito uno Use Case Diagram Generale, che si pone al livello di astrazione più elevato del sistema, mettendo in evidenza tutte quelle che sono le modalità di utilizzo dello stesso, da parte dell'utente e dell'amministratore. Questi ultimi rappresentano gli attori che possono accedere alle funzionalità offerte dal sistema, di cui alcune sono condivise, ossia accessibili da entrambi, mentre altre sono prerogativa esclusiva di uno dei due ruoli.

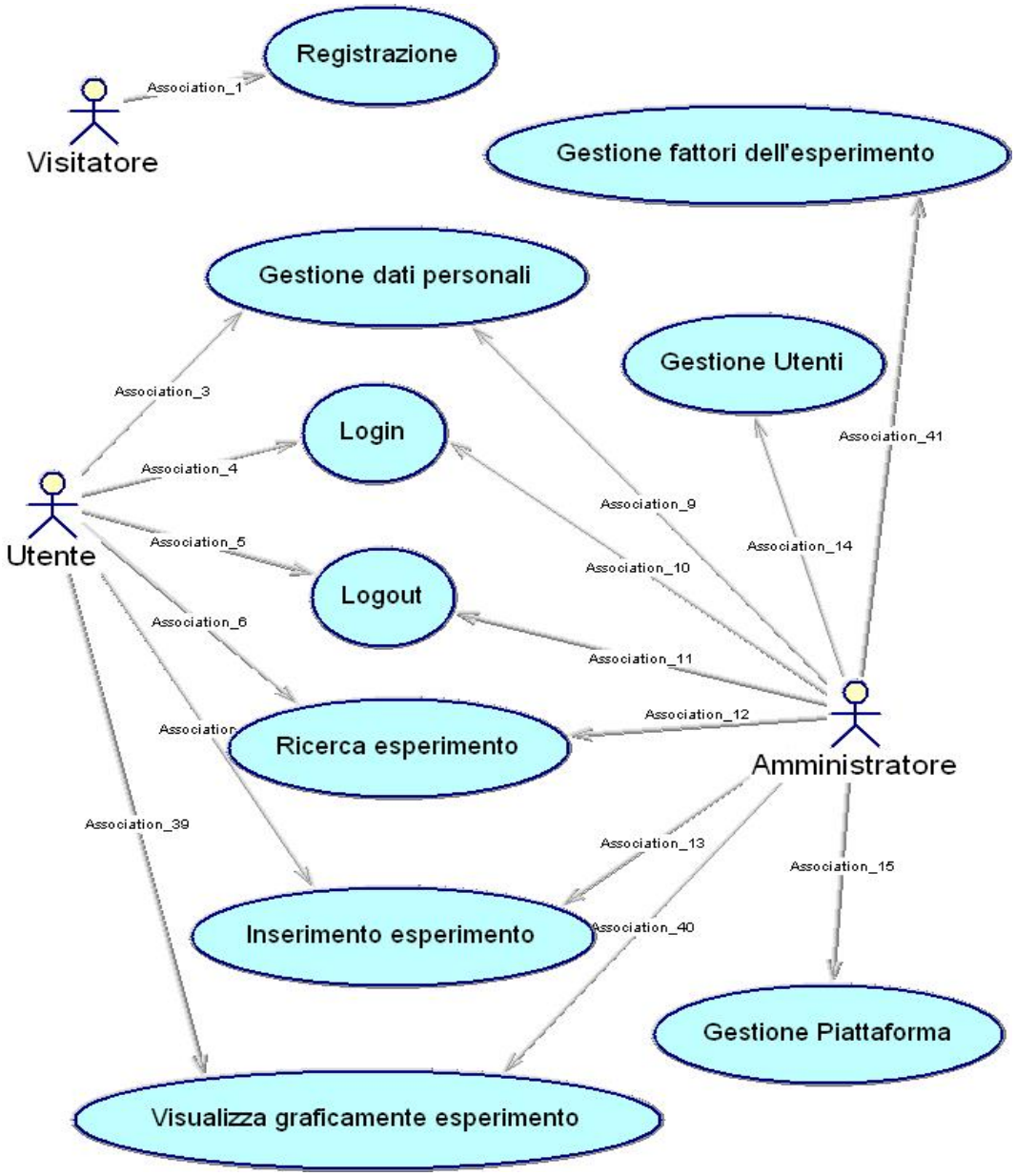


Figura 4.5 – ***Use Case Generale***

4.3.2 Gestione utenti

Considerando nuovamente il diagramma dei casi d'uso Generale, si evince che "Gestione Utenti" comprende alcune azioni basilari che possono essere eseguite sugli utenti registrati, come la visualizzazione di un elenco di questi ultimi, la cancellazione, la visualizzazione dei dati anagrafici di un singolo utente , l'operazione di sblocco, di interdizione,di conferma e di promozione.

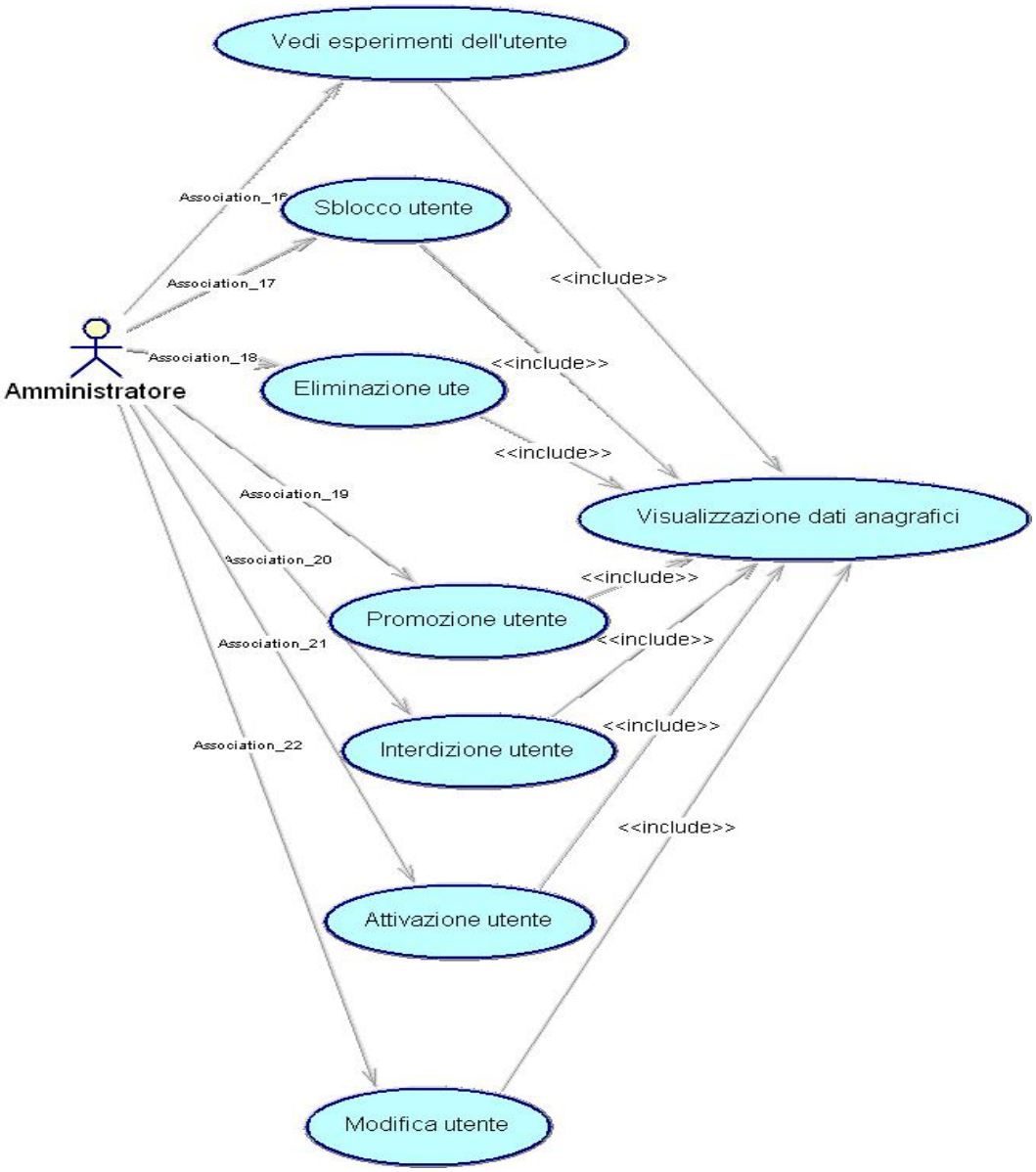


Figura 4.6 – ***Use Case Gestione Utenti***

4.3.3 Gestione dati personali

Questo caso d'uso vede come attori sia l'utente che l'amministratore e comprende le operazioni basilari eseguibili sui dati personali dell'utente, quali la modifica dei dati di accesso(username, password), dei dati anagrafici e la possibilità di richiedere i dati di accesso via email in caso di dimenticanza.

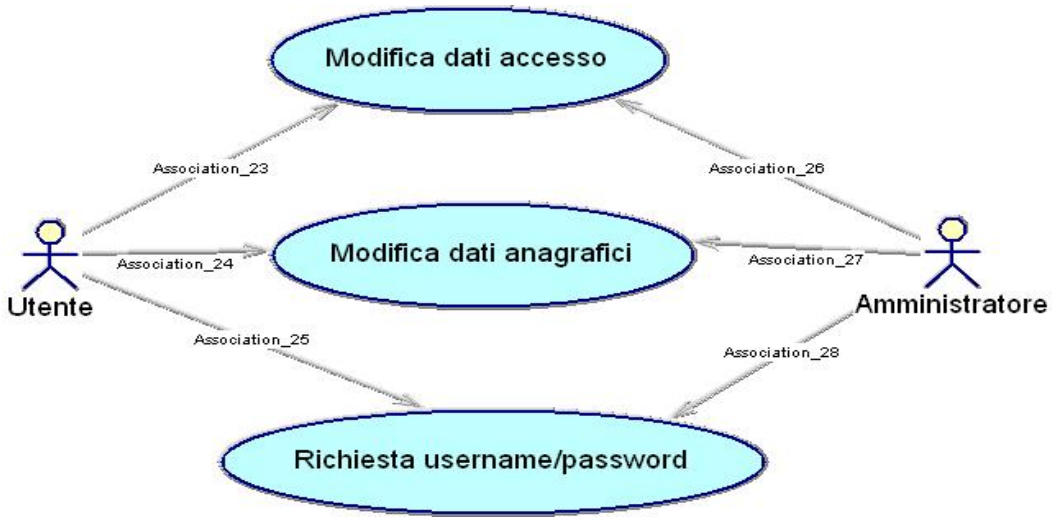


Figura 4.7 – ***Use Case Gestione Dati Personali***

4.3.4 Ricerca

Questo caso d'uso è sicuramente uno dei principali e svolge un compito fondamentale, in quanto racchiude tutte le operazioni realizzabili dall'utente/amministratore nella fase di "navigazione" dell'esperimento, quali particolari ricerche sui dati o visualizzazioni grafiche dei risultati dell'esperimento.

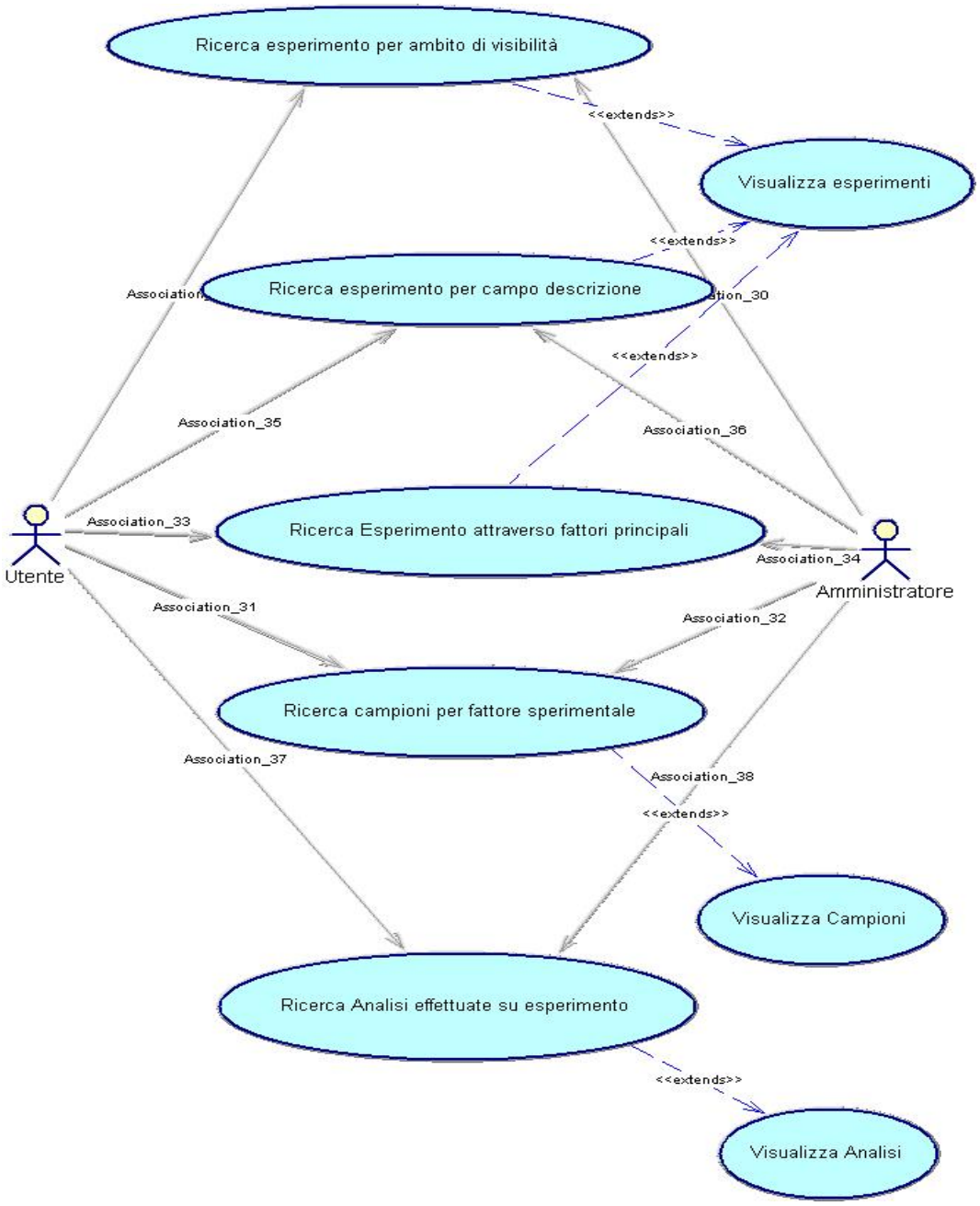


Figura 4.8 – ***Use Case Ricerca***

4.3.5 Gestione fattori dell'esperimento

Questo caso d'uso è a carico esclusivamente dell'amministratore e comprende tutte le operazioni necessarie per la gestione dei dati dell'applicazione, compreso tutti i parametri che entreranno a far parte dell'esperimento nella fase di inserimento di quest'ultimo.

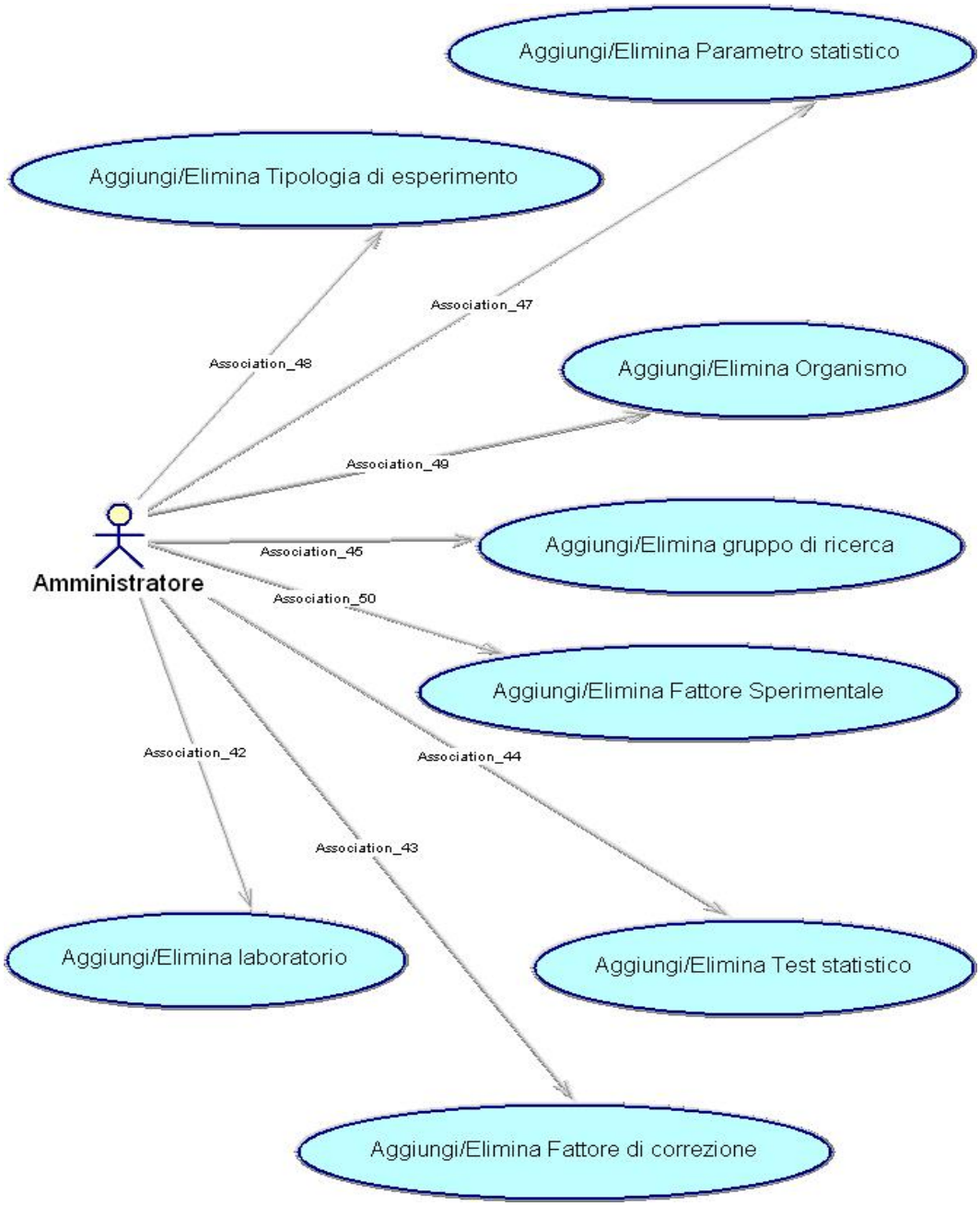


Figura 4.9 – ***Use Case Gestione Fattori dell'esperimento***

4.3.6 Gestione Piattaforme

Questo caso d'uso è a carico esclusivamente dell'amministratore e comprende tutte le operazioni necessarie per la gestione delle piattaforme esistenti nell'applicazione e per l'inserimento di nuove piattaforme nell'archivio.

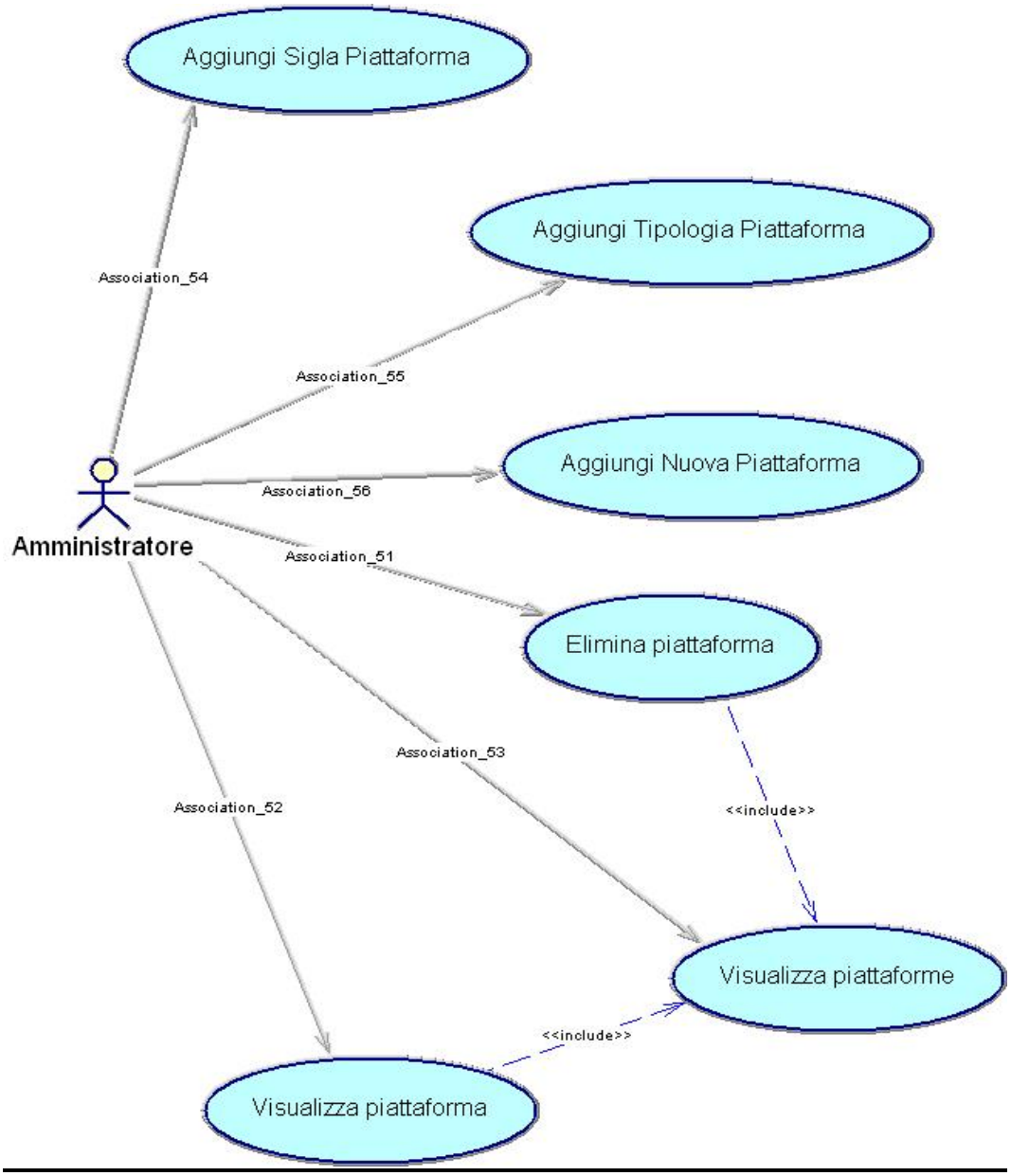


Figura 4.10 – ***Use Case Gestione Piattaforme***

4.3.7 Rappresentazione testuale dei casi d'uso dell'applicazione MicroArray Web

Casi d'uso Visitatore

Descrizione dei Casi d'Uso

Nome**: Registrazione**

Iniziatore: Visitatore

Scopo: Registrare un visitatore all'interno dell'applicazione per usufruirne dei servizi

Scenario principale di successo

1. Il visitatore chiede di essere registrato.
2. Il sistema mostra il Form per la registrazione.
3. Il visitatore compila il Form.
4. Il visitatore trasmette i dati.
5. Il sistema riceve i dati.
6. Il sistema verifica i dati.
7. Il sistema registra i dati.
8. Il sistema comunica l'avvenuta registrazione al visitatore.

Estensioni

6a Campi obbligatori non tutti digitati, formato errato dei dati o lunghezze minime non rispettate.

6b. Username e nome gruppo già inseriti da un altro utente.

6c. E-mail già presente nell'archivio.

6d. Il sistema mostra di nuovo il form all'utente.

Casi d'uso Utente/Amministratore

Nome**: Login**

Iniziatore: Amministratore, utente

Scopo: Autenticazione dell' utente/amministratore

Scenario principale di successo

1. L'utente/amministratore accede al sistema.
2. Il sistema mostra il Form per l'accesso.
3. L'utente/amministratore immette username,password e gruppo di ricerca di appartenenza.
4. L'utente/amministratore trasmette i dati.
5. Il sistema riceve i dati.
6. Il sistema verifica la corrispondenza in archivio dei dati immessi.
7. Il sistema trova una corrispondenza in archivio.
8. Il sistema mostra un messaggio di accesso garantito.
9. Il sistema mostra una vista con i servizi disponibili.

Estensioni

7a. Password non trovata, rimostra il form all'utente, se il numero di tentativi permessi
è >0.

7b. Corrispondenza non trovata. L'utente non è registrato nel sistema

7c. Trovati divieti di accesso per situazioni di interdizione,blocco o attesa dell'utente.

7d. Il sistema mostra di nuovo il form al visitatore.

Nome**: Modifica Username/Password**

Iniziatore: Utente/Amministratore

Scopo: Consente all'utente/amministratore di modificare i propri dati di accesso.

Scenario principale di successo

1. L'utente/amministratore chiede di modificare i propri dati di accesso.
2. **Il sistema mostra all'utente/amministratore la scheda dei propri dati di accesso.**
3. L'utente modifica opportunamente i dati.
4. L'utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema aggiorna i dati di accesso dell'utente/amministratore.
7. Il sistema comunica all'utente/amministratore l'avvenuta operazione di aggiornamento

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'utente.

5b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'utente.

Nome**: Logout**

Iniziatore: Amministratore, utente

Scopo: Logout dell' utente/amministratore

Scenario principale di successo

1. L'utente/amministratore decide di abbandonare il sistema selezionando l'apposita sezione di logout.
2. Il sistema riceve l'id dell'utente/amministratore che ha chiesto il logout.
3. Il sistema esegue il logout, rilasciando le risorse impegnate dall'utente/amministratore.

Nome**: Modifica dati anagrafici**

Iniziatore: Amministratore, utente

Scopo: Consente all'amministratore/utente di modificare i propri dati anagrafici.

Scenario principale di successo

1. L'utente chiede di modificare i propri dati personali.
2. **Il sistema mostra all'utente/amministratore la scheda dei propri dati personali.**
3. L'utente/amministratore modifica opportunamente i dati.
4. L'utente/amministratore invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema aggiorna i dati dell'utente/amministratore.
7. Il sistema comunica all'utente l'avvenuta operazione di aggiornamento

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'utente/amministratore.

5b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'utente/amministratore.

Nome: Richiesta Username/Password

Iniziatore: Amministratore, utente

Scopo: Consente all'amministratore/utente di recuperare i propri dati di accesso in caso di dimenticanza.

Scenario principale di successo

1. L'amministratore/utente chiede di recuperare via email i propri dati di accesso.
2. **Il sistema mostra all'amministratore/utente due form per la digitazione dell'indirizzo e-mail e per la conferma dell'indirizzo.**
3. L'amministratore/utente inserisce opportunamente i dati.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema invia i dati richiesti dall'amministratore/utente all'indirizzo e-mail inserito.
7. Il sistema comunica all'amministratore/utente l'avvenuta operazione di invio dati.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

5b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'amministratore/utente.

Nome**: Inserimento esperimento**

Iniziatore: Amministratore, utente

Scopo: Consente all'amministratore/utente di inserire i dati relativi ad un nuovo esperimento.

Scenario principale di successo

1. L'amministratore/utente chiede di inserire i dati relativi ad un nuovo esperimento.
2. Il sistema mostra all'amministratore/utente svariati forms per l'inserimento dei dati relativi all'esperimento.
3. L'amministratore/utente inserisce opportunamente i dati.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema comunica all'amministratore/utente l'avvenuta archiviazione dei dati relativi all'esperimento.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

5b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'amministratore/utente.

Nome**: Inserimento campione**

Iniziatore: Amministratore, utente

Scopo: Consente all'amministratore/utente di inserire i dati relativi ad un campione analizzato durante l'esperimento.

Scenario principale di successo

1. L'amministratore/utente chiede di inserire i dati relativi ad un nuovo campione.

2. Il sistema mostra all'amministratore/utente svariati forms per l'inserimento dei dati relativi al campione.
3. L'amministratore/utente inserisce opportunamente i dati.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema comunica all'amministratore/utente l'avvenuta archiviazione dei dati relativi al campione selezionato.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

5b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'amministratore/utente.

Nome**: Inserimento risultati esperimento**

Iniziatore: Amministratore,Utente

Scopo: Consente all'amministratore/utente di inserire i risultati relativi all'esperimento in esame.

Scenario principale di successo

1. L'amministratore/utente chiede di inserire i risultati relativi all'esperimento in esame.
2. Il sistema mostra all'amministratore/utente svariati forms per l'inserimento dei risultati.
3. L'amministratore/utente inserisce opportunamente i dati.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema comunica all'amministratore/utente l'avvenuta archiviazione dei risultati relativi all'esperimento in esame.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

5b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'amministratore/utente.

Nome**: Inserimento risultati delle analisi statistiche dell' esperimento**

Iniziatore: Amministratore,Utente

Scopo: Consente all'amministratore/utente di inserire i risultati relativi alle analisi statistiche dell'esperimento in esame.

Scenario principale di successo

1. L'amministratore/utente chiede di inserire i dati relativi alle analisi statistiche sull'esperimento in esame.
2. Il sistema mostra all'amministratore/utente svariati forms per l'inserimento dei risultati delle analisi statistiche.
3. L'amministratore/utente inserisce opportunamente i dati.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema comunica all'amministratore/utente l'avvenuta archiviazione dei risultati relativi alle analisi dell'esperimento in esame.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

5b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'amministratore/utente.

Nome**: Ricerca esperimento per ambito di visibilità**

Iniziatore: Amministratore,Utente

Scopo: Consente all'amministratore/utente di ricercare i dati relativi ad un esperimento.

Scenario principale di successo

L'amministratore/utente chiede di ricercare un esperimento attraverso l'inserimento del suo ambito di visibilità.

Il sistema mostra all'amministratore/utente un form per l'inserimento dell'ambito di visibilità.

L'amministratore/utente inserisce opportunamente i dati.

L'amministratore/utente invia i dati.

Il sistema verifica i dati inseriti.

Il sistema mostra all'amministratore/utente i risultati della ricerca.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

6a. Nessun esperimento trovato rispondente alle scelte dell'utente.

Nome**: Ricerca esperimento attraverso fattori principali**

Iniziatore: Amministratore,Utente

Scopo: Consente all'amministratore/utente di ricercare i dati relativi ad un esperimento.

Scenario principale di successo

1. L'amministratore/utente chiede di ricercare un esperimento attraverso l'inserimento di alcuni suoi parametri.

2. Il sistema mostra all'amministratore/utente 4 forms per l'inserimento del tipo di esperimento, del laboratorio, del fattore sperimentale e dell'autore.

3. L'amministratore/utente inserisce opportunamente i dati.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema mostra all'amministratore/utente i risultati della ricerca.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

6a. Nessun esperimento trovato rispondente alle scelte dell'utente.

Nome**: Ricerca esperimento per campo descrizione**

Iniziatore: Amministratore,Utente

Scopo: Consente all'amministratore/utente di ricercare i dati relativi ad un esperimento.

Scenario principale di successo

L'amministratore/utente chiede di ricercare un esperimento attraverso l'inserimento di parole chiavi relative all'esperimento.

Il sistema mostra all'amministratore/utente un form per l'inserimento del testo da ricercare.

L'amministratore/utente inserisce opportunamente i dati.

L'amministratore/utente invia i dati.

Il sistema verifica i dati inseriti.

Il sistema mostra all'amministratore/utente i risultati della ricerca.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

6a. Nessun esperimento trovato rispondente alle scelte dell'utente.

Nome**: Ricerca campioni per fattore sperimentale**

Iniziatore: Amministratore,Utente

Scopo: Consente all'amministratore/utente di ricercare i dati relativi ai campioni.

Scenario principale di successo

1. L'amministratore/utente chiede di ricercare dei campioni attraverso l'inserimento di un fattore sperimentale e del relativo valore.
2. Il sistema mostra all'amministratore/utente un form per l'inserimento del fattore sperimentale da ricercare ed un form per l'inserimento del relativo valore.
3. L'amministratore/utente inserisce opportunamente i dati.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema mostra all'amministratore/utente i risultati della ricerca.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

6a. Nessun esperimento trovato rispondente alle scelte dell'utente.

Nome**: Ricerca analisi effettuate sull'esperimento**

Iniziatore: Amministratore,Utente

Scopo: Consente all'amministratore/utente di ricercare i dati relativi alle analisi sull'esperimento.

Scenario principale di successo

1. L'amministratore/utente chiede di ricercare le analisi compiute sull'esperimento attraverso l'inserimento dei dati relativi a 5 forms, il campione di prova, quello di riferimento, il test statistico usato, il parametro statistico e il relativo valore.
2. Il sistema mostra all'amministratore/utente un form per l'inserimento del fattore sperimentale da ricercare ed un form per l'inserimento del relativo valore.
3. L'amministratore/utente inserisce opportunamente i dati.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema mostra all'amministratore/utente i risultati della ricerca.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

6a. Nessuna analisi trovata rispondente alle scelte dell'utente.

Nome**: Visualizzazione grafica dei dati normalizzati dell'esperimento**

Iniziatore: Amministratore,Utente

Scopo: Consente all'amministratore/utente di visualizzare graficamente i dati dell'esperimento.

Scenario principale di successo

1. L'amministratore/utente chiede di visualizzare graficamente i dati dell'esperimento.
2. Il sistema mostra all'amministratore/utente la lista degli esperimenti dei quali effettuare la visualizzazione.
3. L'amministratore/utente scegli l'esperimento.
4. L'amministratore/utente invia i dati.
5. Il sistema verifica i dati inseriti.
6. Il sistema mostra all'amministratore/utente i risultati della ricerca.

Estensioni

5a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'amministratore/utente.

Casi d'uso Amministratore

Nome: Gestione Utenti

Iniziatore: Amministratore

Scopo: Alcune azioni basilari che possono essere eseguite sugli utenti registrati, in particolare la visualizzazione di un elenco di questi ultimi, la cancellazione, la visualizzazione dei dati anagrafici di un singolo utente e l'operazione di sblocco e di interdizione di un utente.

Nome: : Visualizzazione utenti

Iniziatore: Amministratore

Scopo: Consente all'amministratore di visualizzare tutti gli utenti registrati nell'applicazione con i relativi dati anagrafici e di accesso.

Scenario principale di successo

1. L'amministratore chiede di visualizzare utenti e relativi dati.
2. Il sistema mostra tutti gli utenti iscritti all'applicazione con i relativi dati.

Estensioni

2a. Nessun utente iscritto, viene rimostrato all'amministratore il menu di selezione.

Nome: **Interdizione utente**

Iniziatore: Amministratore

Scopo: Consente all'amministratore di interdire un utente sospendendolo dall'usufruire delle funzionalità dell'applicazione

Scenario principale di successo

1. L'amministratore chiede di interdire un utente.
2. Il sistema mostra tutti gli utenti iscritti all'applicazione.
3. L'amministratore sceglie l'utente da interdire.
4. L'amministratore trasmette i dati.
5. Il sistema riceve i dati.
6. Il sistema verifica i dati.
7. Il sistema registra le modifiche sull'utente e pone l'utente in stato interdetto.
8. Il sistema comunica all'amministratore l'avvenuta interdizione dell'utente.

Nome: Cancellazione utente

Iniziatore: Amministratore

Scopo: Elimina un utente dall'applicazione

Scenario principale di successo

1. L'amministratore chiede di eliminare un utente dall'applicazione.
2. Il sistema mostra tutti gli utenti iscritti all'applicazione.
3. L'amministratore sceglie l'utente da eliminare.
4. L'amministratore trasmette i dati.
5. Il sistema riceve i dati.
6. Il sistema verifica i dati.
7. Il sistema elimina l'utente dall'applicazione.
8. Il sistema comunica all'amministratore l'avvenuta eliminazione dell'utente.

Nome**: Sblocco utente**

Iniziatore: Amministratore

Scopo: Sblocco di un utente dall'interdizione

Scenario principale di successo

1. L'amministratore chiede di sbloccare un utente precedentemente interdetto.
2. Il sistema mostra tutti gli utenti iscritti all'applicazione.
3. L'amministratore sceglie l'utente da sbloccare.
4. L'amministratore trasmette i dati.
5. Il sistema riceve i dati.
6. Il sistema verifica i dati.
7. Il sistema sblocca l'utente.
8. Il sistema comunica all'amministratore l'avvenuto sblocco dell'utente.

Nome**: Promozione Utente**

Iniziatore: Amministratore

Scopo: Promozione di un utente ad amministratore

Scenario principale di successo

L'amministratore chiede di promuovere un utente ad amministratore.

Il sistema mostra tutti gli utenti iscritti all'applicazione.

L'amministratore sceglie l'utente da promuovere.

L'amministratore trasmette i dati.

Il sistema riceve i dati.

Il sistema verifica i dati.

Il sistema promuove l'utente.

Il sistema comunica all'amministratore l'avvenuta promozione dell'utente.

Nome: Gestione Piattaforme

Iniziatore: Amministratore

Nome: Inserisci piattaforma

Iniziatore: Amministratore

Scopo: Inserire una nuova piattaforma nel sistema

Scenario principale di successo

1. L'amministratore chiede di inserire una nuova piattaforma nel sistema.
2. Il sistema mostra un form per l'inserimento dei dati sulla piattaforma.
3. L'amministratore compila il form.
4. L'amministratore trasmette i dati.
5. Il sistema riceve i dati.

6. Il sistema verifica i dati.
7. Il sistema aggiunge la piattaforma all'elenco delle piattaforme del sistema.
8. Il sistema comunica all'amministratore l'avvenuto inserimento della nuova piattaforma nel sistema.

Estensioni

6a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'utente.

6b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'utente.

Nome**: Modifica dati piattaforma**

Iniziatore: Amministratore

Scopo: Modificare i dati di una piattaforma nel sistema

Scenario principale di successo

1. L'amministratore chiede di modificare i dati di una piattaforma nel sistema.
2. Il sistema mostra mostra tutte le piattaforme presenti nel sistema.
3. L'amministratore sceglie la piattaforma da modificare.
4. L'amministratore trasmette i dati.
5. Il sistema riceve i dati.
6. Il sistema verifica i dati.
7. Il sistema mostra i dati della piattaforma per le modifiche.
8. L'amministratore effettua le modifiche.
9. L'amministratore invia le modifiche.
10. Il sistema comunica all'amministratore le avvenute modifiche alla piattaforma.

Estensioni

6a. Alcuni campi obbligatori non sono stati inseriti, il sistema rimostra il form all'utente.

6b. Alcuni campi non presentano valori corretti nel formato,il sistema rimostra il form all'utente.

Nome**: Elimina piattaforma**

Iniziatore: Amministratore

Scopo: Eliminare una piattaforma dal sistema

Scenario principale di successo

1. L'amministratore chiede di eliminare una piattaforma dall'applicazione.
2. Il sistema mostra all'amministratore l'elenco delle piattaforme disponibile.
3. L'amministratore sceglie la piattaforma da eliminare.
4. L'amministratore invia i dati.
5. Il sistema elimina la piattaforma.
6. Il sistema comunica all'amministratore l'avvenuta eliminazione della piattaforma dal sistema.

4.4 Activity Diagrams

Facendo riferimento a quelle che sono le funzionalità offerte dal sistema e alle modalità di utilizzo di quest'ultimo, attraverso gli Activity Diagrams è possibile modellare le azioni necessarie per compiere un'attività ed il flusso di controllo tra di esse.

Ciascuna "activity" rappresenta un'azione che viene eseguita dal sistema oppure dall'utilizzatore e può essere atomica oppure scomposta in più "action" elementari.

Nella gestione del flusso di controllo è possibile definire esecuzioni parallele di operazioni e la loro sincronizzazione, nonché esecuzioni condizionali.

Inoltre, attraverso lo strumento dell' "object flow", è possibile specificare quali siano gli oggetti del sistema che entrano in gioco per ciascuna azione eseguita e quale sia il loro stato.

E' stato definito un Activity Diagram di carattere generale, che si pone ad un elevato livello di astrazione e che descrive tutte le possibili azioni ed il relativo flusso, che possono essere eseguite da un utente non registrato, da un utente registrato ed ovviamente dall'amministratore.

4.4.1 Activity diagram generale

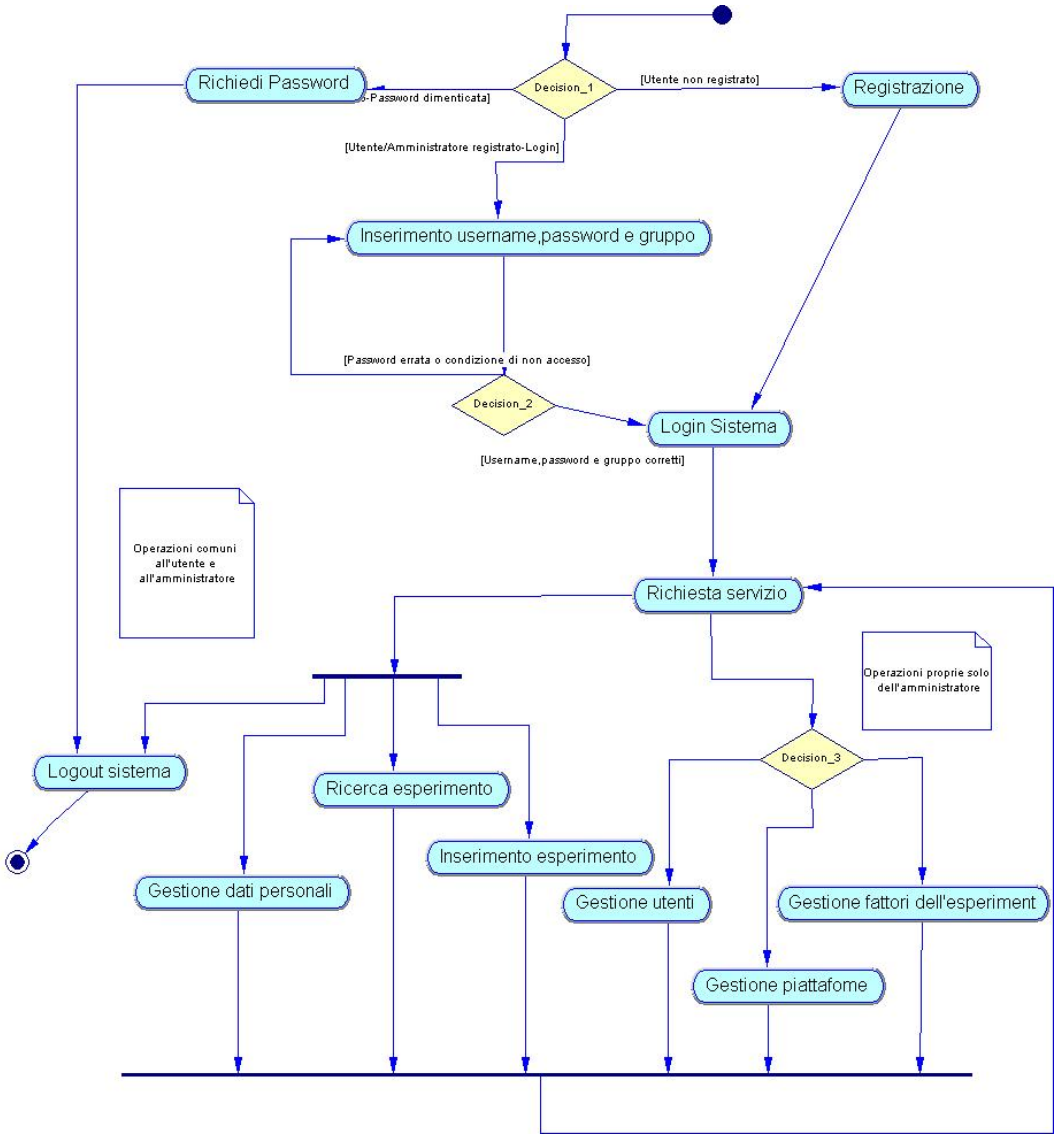


Figura 4.11 – ***Activity Diagram Generale***

A partire da esso, è possibile scendere ad un livello di dettaglio maggiore, specificando la sequenza delle azioni per ogni funzionalità del sistema.

4.4.2 Login

Per quanto riguarda il Login dell'utente e dell'amministratore, le azioni da eseguire sono praticamente le stesse a meno degli oggetti che entrano in gioco.

Una volta inseriti Username, Password e nome del gruppo di ricerca, il sistema ne effettua il controllo della correttezza, valutando in primo luogo se la coppia username-nomegruppo esiste e successivamente verificando la validità della Password.

Ovviamente, a colui che tenta di accedere viene sempre dato un messaggio generico di errore, in particolare, in caso di errore sulla password questo viene comunicato e viene decrementato un contatore degli accessi possibili all'utente.

Superati i 3 errori consecutivi l'utente viene interdetto, inserendo la tripla di valori corretta il contatore dell'utente è ripristinato al valore iniziale 3.

Per evitare tentativi di accesso non autorizzati, dopo che l'utente si è registrato, il sistema blocca tale account e lo lascia in uno stato di attesa di conferma da parte dell'amministratore.

Ovviamente, nel caso in cui Username, Password e nome gruppo siano corretti, si valuta

comunque se è attivo un blocco sull'account; nel caso di login consentito, si esegue l'aggiornamento delle informazioni di accesso (ripristino del contatore a 3,se necessario).

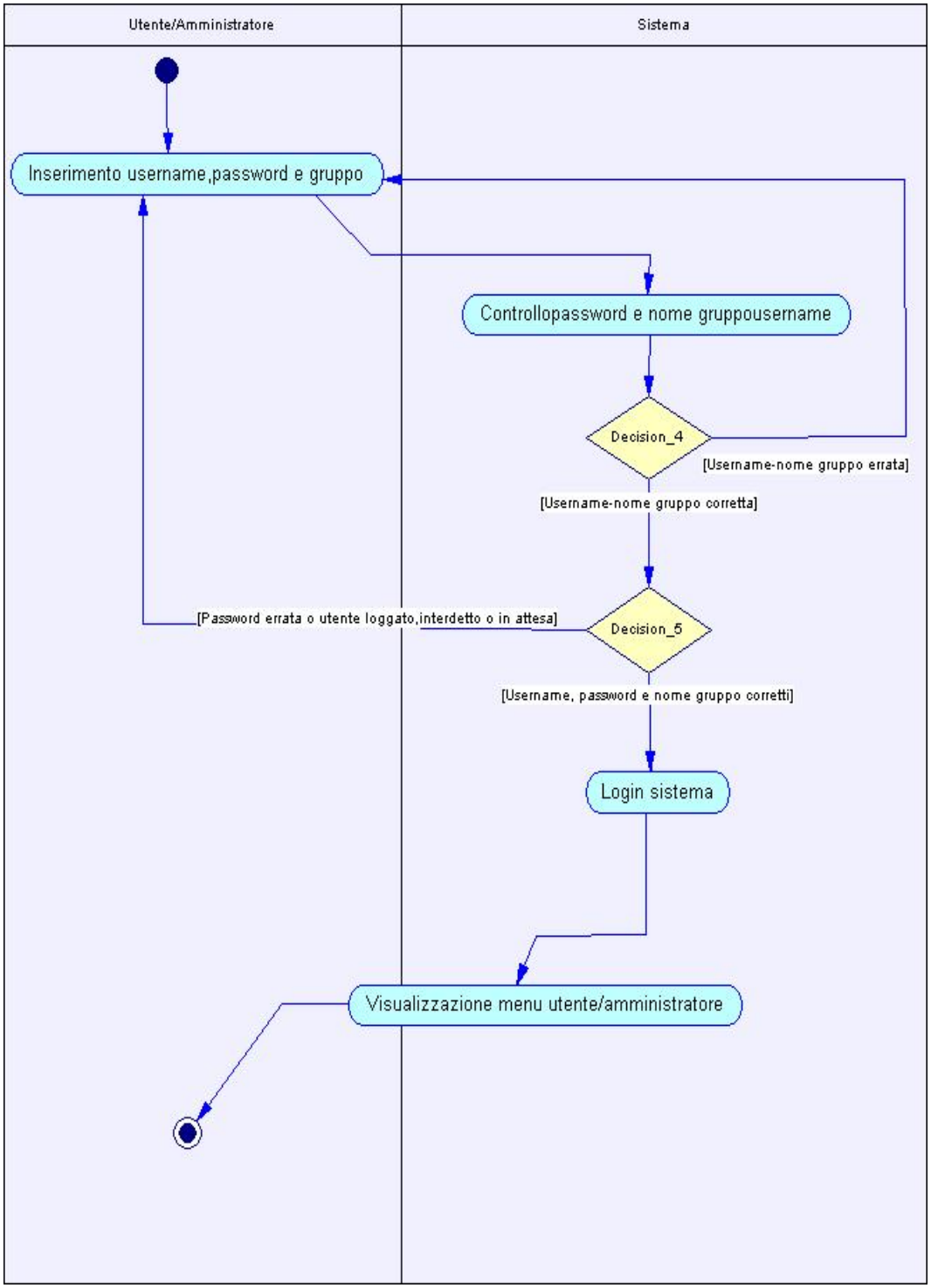


Figura 4.12– ***Activity Diagram Login***

4.4.3 Registrazione

La registrazione dell'utente per poter usufruire dei servizi della Web application prevede l'immissione da parte dell'utente di:

1. dati di accesso, Username, Password e nome gruppo di ricerca.
2. dell'indirizzo email e dei propri dati anagrafici.

Ovviamente, sono previste tutte le operazioni di validazione dei dati, necessarie a garantire che i dati immessi dall'utente rispettino i formati attesi dall'applicazione.

Al termine della registrazione tale account viene messo in stato di attesa di conferma da parte dell'amministratore.

Tale account resterà bloccato finché l'amministratore non provvederà a sbloccarlo e l'utente potrà così automaticamente accedere all'applicazione.

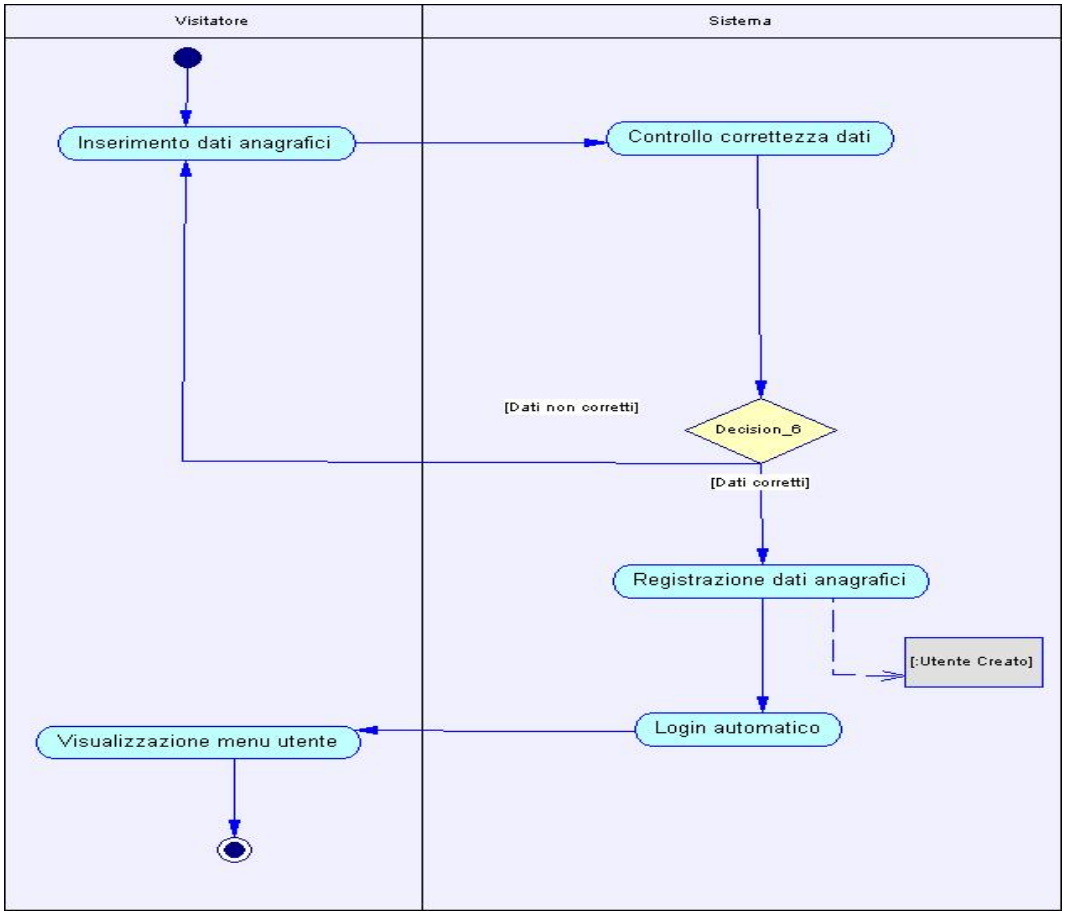


Figura 4.13– ***Activity Diagram Registrazione***

4.4.4 Richiesta Username/Password

Nel caso in cui un utente registrato abbia perso oppure dimenticato le informazioni relative all'accesso, è possibile utilizzare la funzionalità del sistema che permette all'utente di ricevere una mail contenente tali informazioni.

L'utente deve fornire al sistema l'indirizzo email con cui ha effettuato la registrazione, dopodiché il sistema stesso eseguirà il controllo di correttezza del formato dell'indirizzo e la verifica che quest'ultimo sia correttamente registrato.

In caso di esito positivo, l'utente riceverà una mail con tutte le informazioni richieste.

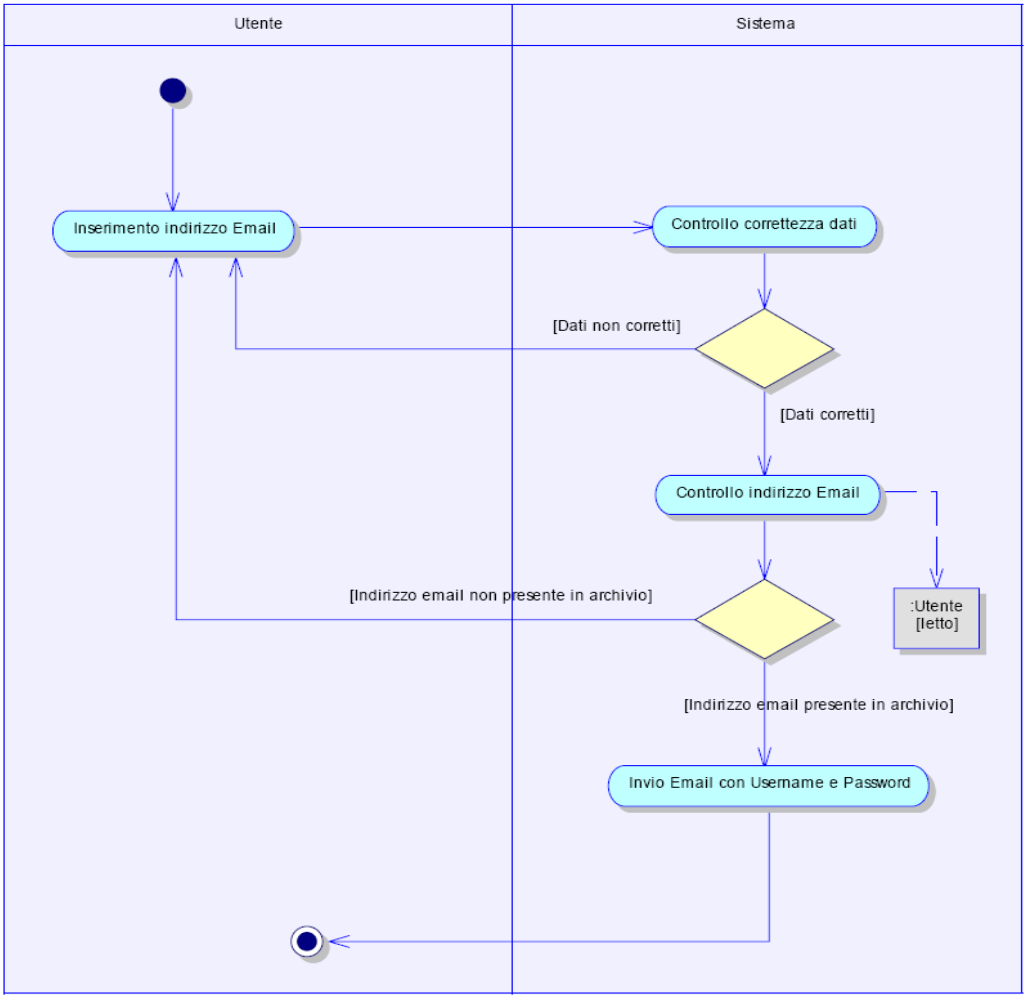


Figura 4.14– *Activity Diagram Richiesta Username/Password*

4.4.5 Modifica dati personali

Nel caso in cui un utente registrato o l'amministratore decidano di modificare i loro dati di accesso o quelli anagrafici, è possibile utilizzare la funzionalità del sistema che permette all'utente/amministratore di modificare questi ultimi.

Il sistema effettuerà dei controlli sui dati vecchi e sui nuovi proposti dall'utente/amministratore e nel caso in cui non esistano condizioni di dati già in uso provvederà all'aggiornamento degli stessi.

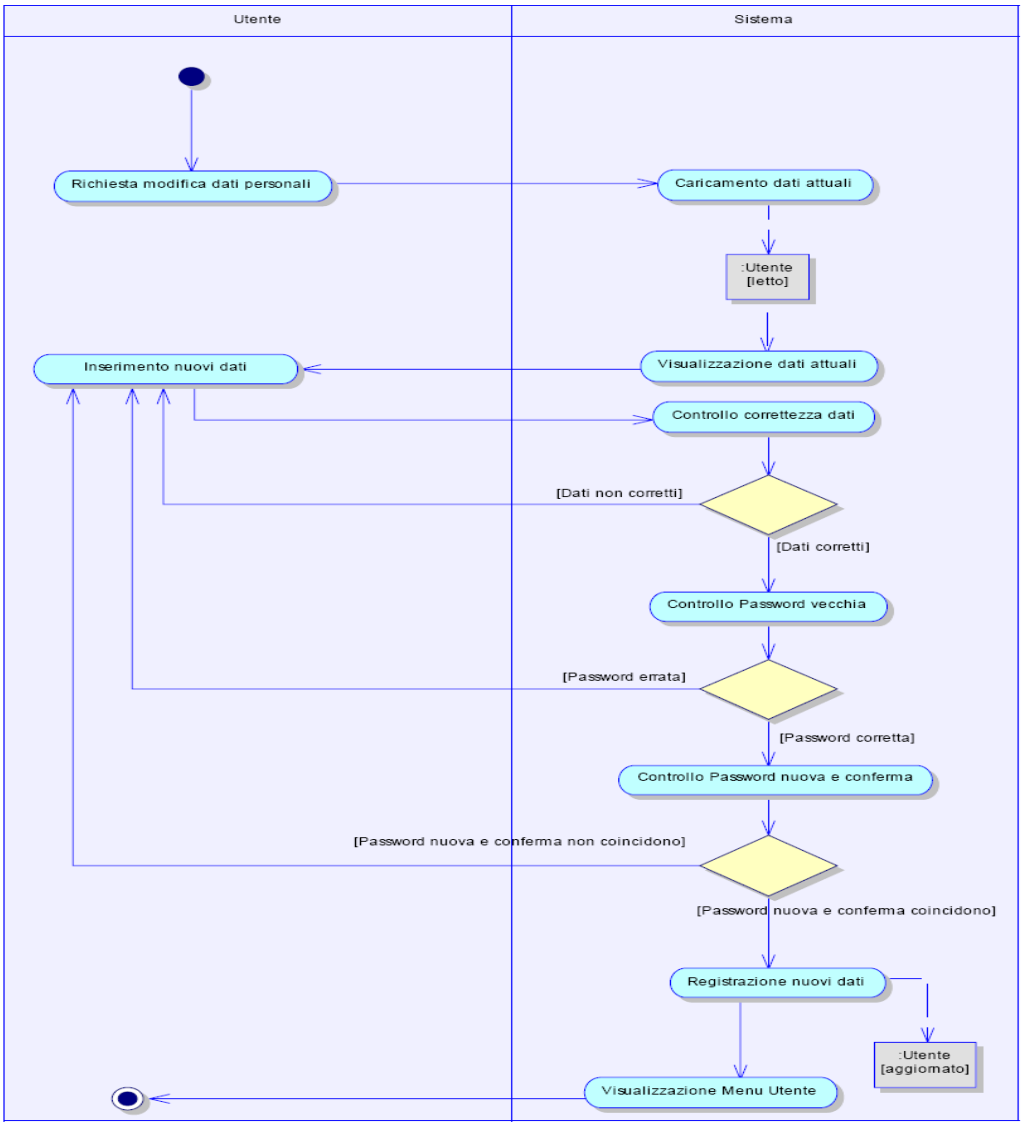


Figura 4.15– ***Activity Diagram Modifica Dati Personali***

4.4.6 Logout

L'ultimo diagramma delle attività che andiamo a presentare riguarda la fase del logout, nella quale l'utente/amministratore richiede l'operazione e il sistema la esegue aggiornando il database utente.

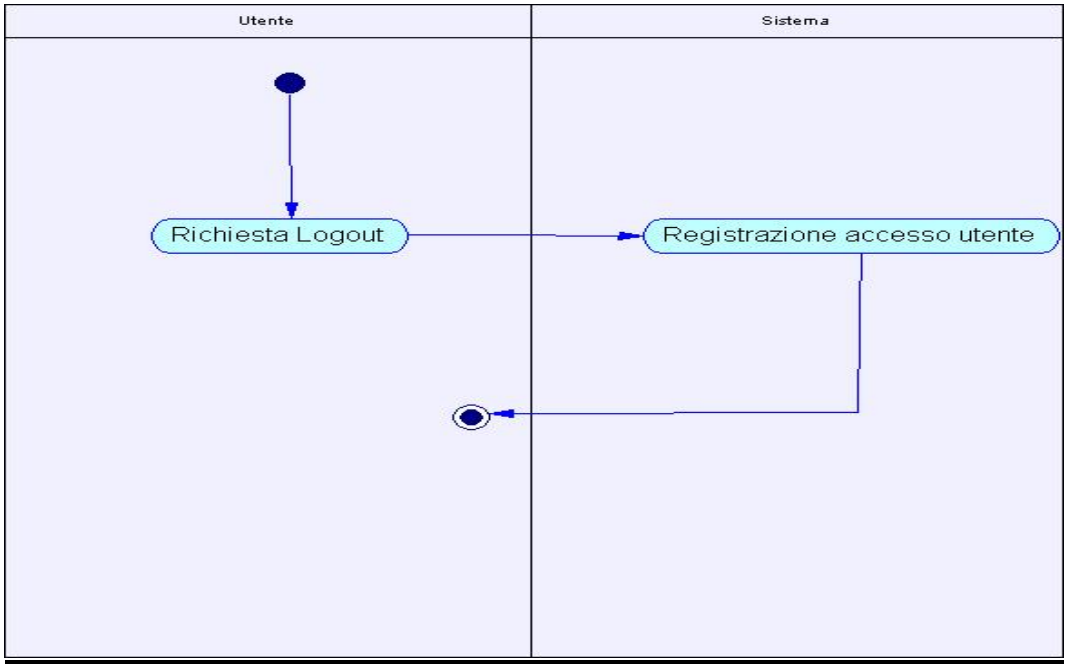


Figura 4.16– ***Activity Diagram Logout***

4.5 Sequence Diagrams

Come visto in precedenza, con i Class Diagrams è possibile individuare le interazioni tra le pagine e gli oggetti. Un altro aspetto molto importante nello sviluppo di una applicazione Web è rappresentato dallo studio delle operazioni da effettuare da un punto di vista dinamico, e non statico come avviene con i class diagram.

A tal proposito si procede alla realizzazione dei Sequence Diagrams, o diagrammi di sequenza, probabilmente i diagrammi più importanti ai fini della realizzazione di una applicazione Web.

Attraverso i Sequence Diagrams è possibile descrivere la vista dinamica del sistema, enfatizzando le interazioni dovute allo scambio di messaggi tra gli oggetti e il loro ordinamento temporale. Ciascun diagramma evidenzia il modo in cui uno scenario, ossia uno specifico percorso in un caso d'uso, viene risolto dalla collaborazione tra un insieme di oggetti.

Generalmente, però, al posto di realizzare un diagramma per ciascuno scenario, si preferisce definire un unico diagramma all'interno del quale vengono descritte le situazioni alternative, facendo uso dei numeri di sequenza sui messaggi.

E' da osservare che nell'ambito dei Sequence Diagrams sono previsti degli oggetti di tipo "Interfaccia", che definiscono le parti del sistema che interagiscono con l'utente, ossia la cosiddetta UI (User Interface). In questa fase non viene specificata la tipologia di interfaccia utente, in modo da poter sfruttare i diagrammi realizzati anche con implementazioni diverse.

E' ovvio come, nel caso di una Web application, l'interfaccia utente sia caratterizzata da pagine Web e dai relativi form che le compongono.

La nostra applicazione web, pur presentando una notevole varietà di funzioni, ha una struttura molto semplice per cui le operazioni fondamentali che l'utente potrà eseguire sono molto elementari e di diagrammi del genere è difficile apprezzarne l'importanza, ma ne mostreremo comunque alcuni per completezza di trattazione.

4.5.1 Registrazione

Per quanto riguarda il Sequence Diagram, si evidenzia il possibile scenario relativo alla registrazione dell'utente.

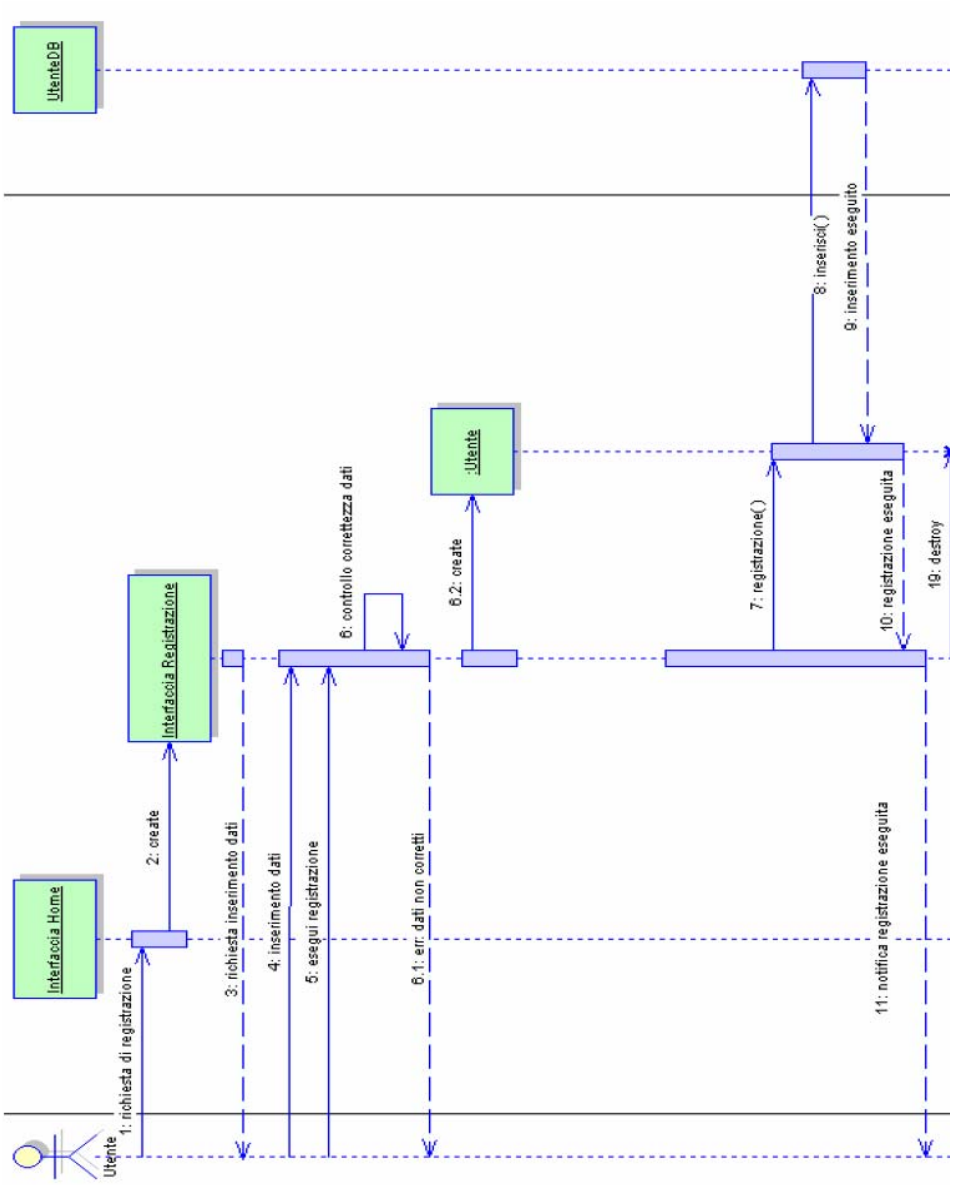


Figura 4.17– *Sequenze Diagram Registrazione*

4.5.2 Modifica dati personali

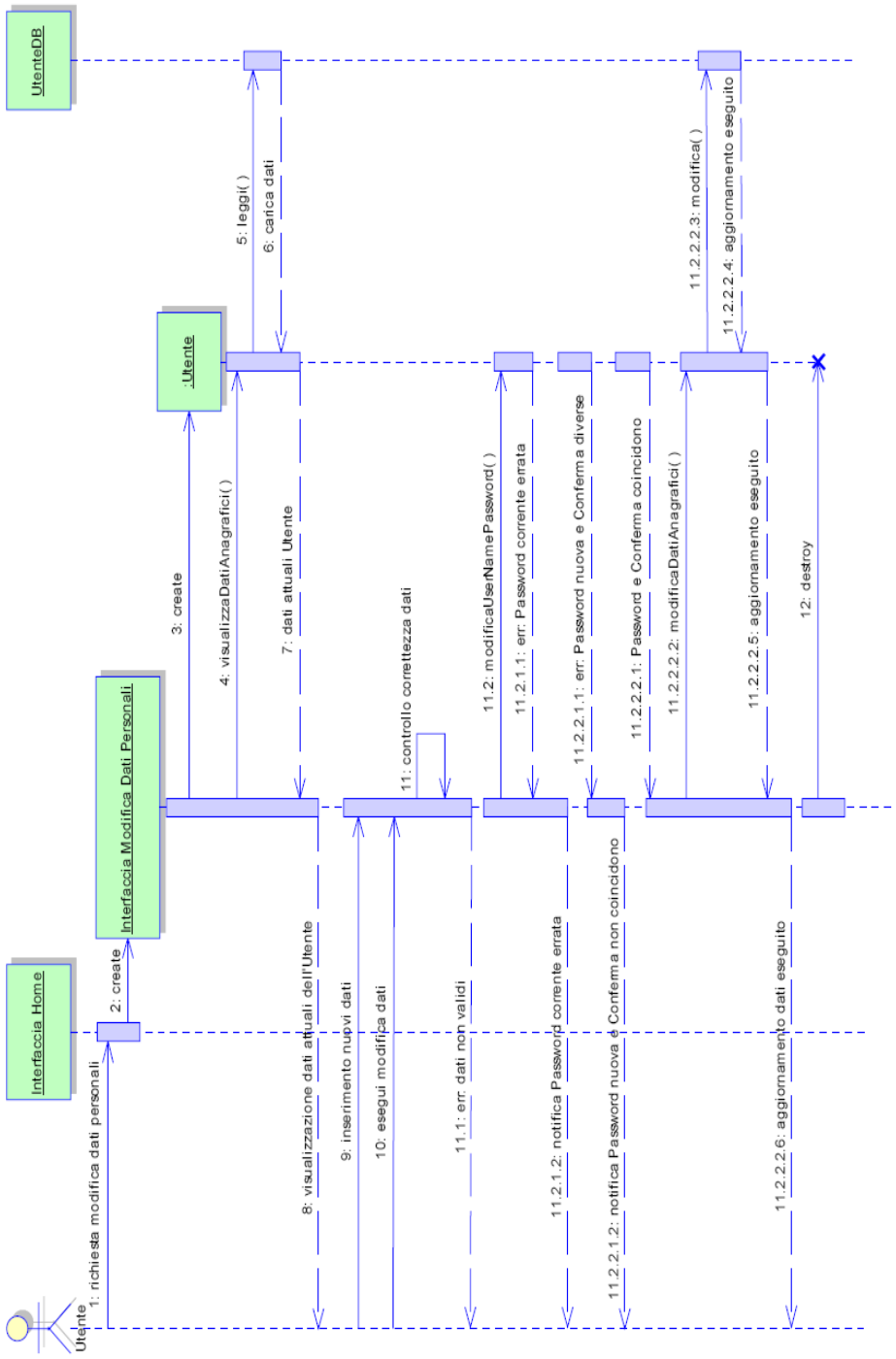


Figura 4.18– ***Sequence Diagram Modifica Dati Personali***

4.5.3 Richiesta username/password

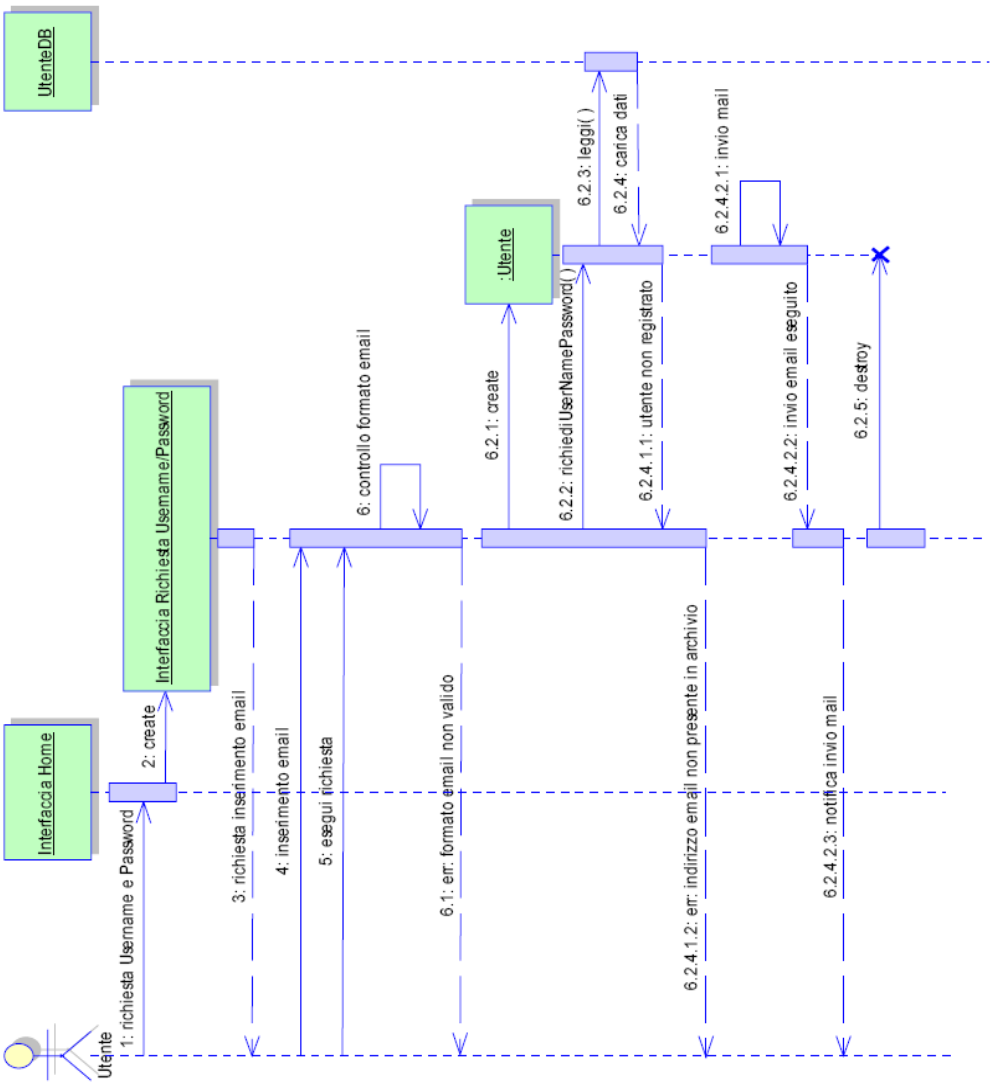


Figura 4.19– ***Sequence Diagram Richiesta Username/Password***

4.5.4 Logout

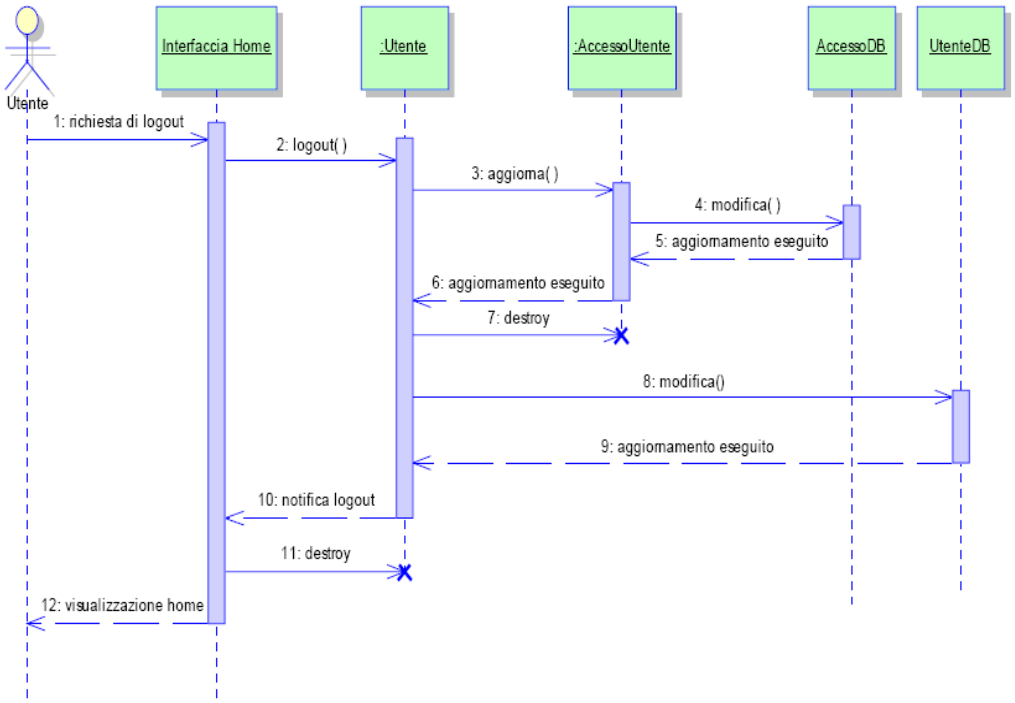


Figura 4.20– *Sequence Diagram Logout*

4.6 Progetto del database Microarray

4.6.1 Costruzione del Modello Concettuale

La definizione del modello concettuale rappresenta il primo passo verso la progettazione della base di dati. Esso permette di descrivere tutte quelle che sono le entità del modello reale e le relazioni tra di esse. Per ciascuna entità o relazione è possibile definirne gli attributi, ponendo particolare attenzione ai cosiddetti attributi primari delle entità, che sono quelli che in seguito diventeranno le "chiavi primarie" delle tabelle finali.

In un certo senso lo stesso Class Diagram può essere considerato,in moltissimi casi, un modello concettuale, poiché le entità e le relazioni rappresentate sono praticamente le stesse, a meno di un formalismo differente.

Per quanto riguarda il nostro problema in esame, possiamo riassumere in questa maniera i passi principali che lo descrivono(facilitando la comprensione del modello concettuale che andremo a mostrare).

Un esperimento realizzato con la tecnica del microarray può essere realizzato su diversi array.

Ogni esperimento consta di una parte "teorica" o "centrale", comune a tutto l'esperimento, ed una parte riguardante i campioni analizzati e le analisi statistiche sui dati, differente per ogni campione e per ogni analisi.

In un esperimento vengono analizzati uno o più campioni e sui dati dell'esperimento possono essere fatte 1 o più analisi. La descrizione di un esperimento si basa sull'utilizzo di diversi fattori ed ogni esperimento può essere realizzato da un utente che appartiene ad uno dei gruppi di ricerca memorizzati nell'applicazione.

E' ora possibile costruire il modello concettuale, o modello E-R, realizzato con il CASE PowerDesigner :

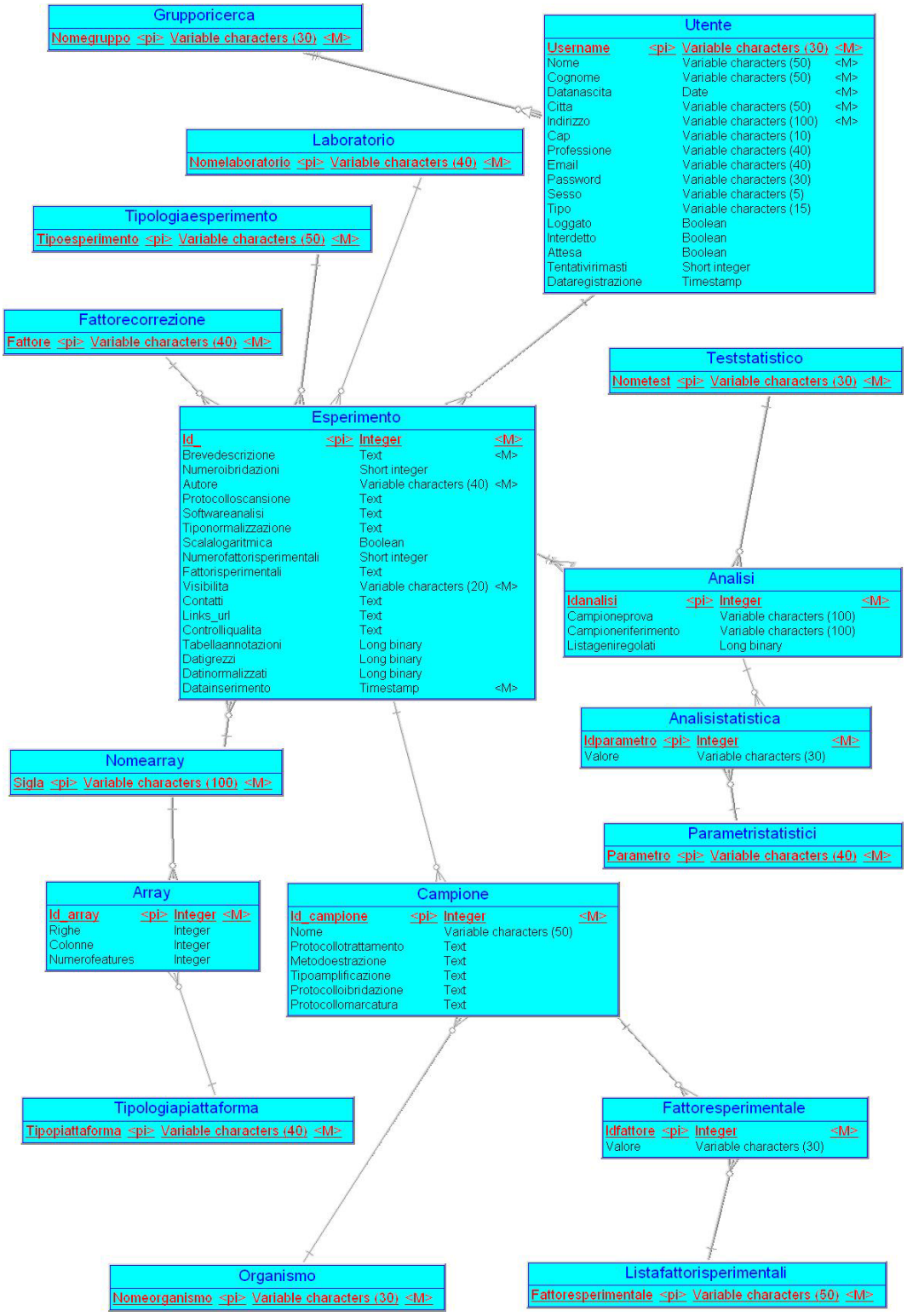


Figura 4.21 – ***Conceptual Model***

4.6.2 Dizionario dei dati

Presentiamo ora delle piccole schede delle singole entità. Per ognuna si dà una breve descrizione e, laddove necessario, delle spiegazioni aggiuntive sugli attributi.

Poiché la maggior parte delle entità sono entità elementari e atomiche ci soffermeremo su quelle fondamentali su cui si incentra il database, essendo le altre comprensibili senza ulteriori chiarimenti.

Gruppo Ricerca

Grupporicerca			
Nomegruppo	<pi>	Variable characters (30)	<M>

Rappresenta un gruppo di ricerca per l'applicazione al quale possono far capo uno o più utenti.

Utente

Utente			
Username	<pi>	Variable characters (30)	<M>
Nome		Variable characters (50)	<M>
Cognome		Variable characters (50)	<M>
Datanascita		Date	<M>
Citta		Variable characters (50)	<M>
Indirizzo		Variable characters (100)	<M>
Cap		Variable characters (10)	
Professione		Variable characters (40)	
Email		Variable characters (40)	<M>
Password		Variable characters (30)	<M>
Sesso		Variable characters (5)	<M>
Tipo		Variable characters (15)	<M>
Loggato		Boolean	<M>
Interdetto		Boolean	<M>
Attesa		Boolean	<M>
Tentativirimasti		Short integer	
Dataregistrazione		Timestamp	<M>

Rappresenta l'utilizzatore del nostro sistema. Ha delle informazioni necessarie alla sua identificazione nell'applicazione,come la username e la password ed informazioni facoltative.

E' legato da una relazione di molti ad uno all'entità gruppo di ricerca e di uno a molti all'entità esperimento, in quanto lo stesso utente può inserire più esperimenti ma un esperimento deve fare capo ad un solo utente.

Esperimento

Esperimento			
Id	<pi>	Integer	<M>
Brevedescrizione		Text	<M>
Numeroibridazioni		Short integer	
Autore		Variable characters (40)	<M>
Protocolloscansione		Text	
Softwareanalisi		Text	
Tiponormalizzazione		Text	
Scalalogaritmica		Boolean	
Numerofattorisperimentali		Short integer	
Fattorisperimentali		Text	
Visibilita		Variable characters (20)	<M>
Contatti		Text	
Links_url		Text	
Controlliqualita		Text	
Tabellaannotazioni		Long binary	
Datigrezzi		Long binary	
Datinormalizzati		Long binary	
Datainserimento		Timestamp	<M>

E' l'entità centrale del nostro database. Un esperimento ha una notevole varietà di attributi, relazionati ad esso la maggior parte da relazioni di uno a molti.

Una relazione molto importante è quella di uno a molti con l'entità analisi, in quanto ad un esperimento possono corrispondere una o più analisi statistiche dei dati.

Inoltre una relazione fondamentale è quella di uno a molti con l'entità campione in quanto in ogni esperimento si analizzeranno uno o più campioni sotto particolari condizioni.

Per quanto riguarda i dati degli esperimenti, notiamo che questi ultimi vengono inseriti in un campo di tipo long binary, che prevede in ingresso un array di byte.

In particolare i dati grezzi e la tabella delle annotazioni non prevedono elaborazioni da parte del sistema, per cui i relativi dati saranno inseriti in un formato .zip e trasformati dall'applicazione in un array di byte.

Per quanto riguarda i dati normalizzati, abbiamo fornito l'applicazione di alcune routines per la lettura del formato di questi dati, che verranno inseriti sì come array di byte, ma secondo un nostro formato interno, per far sì che possano essere ricodificati nel formato originale quando sarà necessario all'applicazione (durante l'uso di Vimida e per la visualizzazione del file).

Campione

Campione			
Id_campione	<pi>	Integer	<M>
Nome		Variable characters (50)	
Protocollotrattamento		Text	
Metodoestrazione		Text	
Tipoamplificazione		Text	
Protocolloibridazione		Text	
Protocollomarcatura		Text	

In un singolo esperimento possono essere analizzati uno o più campioni. Ogni campione ha un Nome significativo che serve a descrivere in maniera chiara ed univoca il campione e risulterà utile nella fase di analisi delle singole liste di geni regolati, per capire da quali coppie di campioni sono ottenute.

Un campione può essere analizzato sulla base di diversi fattori sperimentali, per questo esiste una fondamentale relazione di uno a molti tra l'entità campione e quella fattoresperimentale.

Analisi

Analisi			
Idanalisi	<pi>	Integer	<M>
Campioneprova		Variable characters (100)	
Campioneriferimento		Variable characters (100)	
Listageniregolati		Long binary	

Da un esperimento realizzato con la tecnica del microarray possono essere effettuate diverse analisi dei dati, osservando i campioni inseriti a coppie, con determinati parametri statistici e determinati valori. Ogni analisi del genere restituisce una diversa lista contenente i geni regolati sotto quelle particolari condizioni.

4.6.3 Vincoli e valori di default

Si specificano ora tutti i vincoli, non rilevabili direttamente dal diagramma o comunque non specificati nella descrizione delle entità, che i valori associati agli attributi devono soddisfare.

In particolare, per quanto riguarda l'entità utente, l'attributo tentativirimasti è inizializzato al valore 3, mentre i booleani loggato e interdetto sono inizializzati a falso.

Il tipo dell'utente è inizializzato a "utente" mentre attesa è inizializzato a 1(vero), per rappresentare la condizione che alla creazione di un utente quest'ultimo è posto in attesa della conferma da parte dell'amministratore.

4.6.4 Analisi delle Ridondanze

Dall'analisi del modello concettuale non risultano ridondanze, per cui non è necessario adottare particolari soluzioni.

Va ricordato che la decisione di mantenere o eliminare una ridondanza va presa analizzando i vantaggi e gli svantaggi in termini di accessi in memoria e di occupazione di memoria con o senza il dato ridondante.

4.6.5 Costruzione del modello fisico dei dati

Per completare la progettazione della base di dati è necessario effettuare la trasformazione del modello concettuale nel modello logico, anche detto modello fisico. Quest'ultimo è costituito da una serie di tabelle e dalle relative colonne che prendono il posto delle entità e delle relazioni con i loro attributi, propri del modello concettuale.

La trasformazione viene eseguita sulla base di una serie di regole che permettono di adattare il mondo reale, rappresentato dal modello concettuale, all'implementazione fisica che verrà successivamente adottata per uno specifico DBMS(DataBase Management System).

Le principali operazioni di trasformazione sono le seguenti:

- *eliminare tutte le relazioni del tipo* *Generalizzazione/Specializzazione*, sulla base di tre possibili alternative:
  1. accorpamento delle entità figlie nel padre, con l'introduzione di un attributo identificativo.
  2. accorpamento del padre nelle entità figlie.
  3. sostituzione delle generalizzazioni con semplici associazioni.

- *Scegliere uno o più attributi* di ciascuna entità che possano costituirne una *chiave primaria*, ossia tale da individuare ogni istanza dell'entità stessa in maniera univoca.

  Nel caso in cui l'entità non sia caratterizzata da attributi con questo tipo di funzionalità è possibile introdurre un nuovo attributo che abbia il solo ruolo di chiave primaria.

- *Ogni entità va rappresentata con una tabella*, le cui colonne sono esattamente gli attributi dell'entità stessa;

- Una relazione uno-a-molti tra due entità viene rappresentata con una tabella ma viene assorbita dall'entità lato "uno";

- Una relazione molti-a-molti viene rappresentata attraverso una tabella che sarà legata alle due tabelle relative alle entità in gioco, sulla base delle loro chiavi primarie.

- Una relazione uno-a-uno non viene rappresentata attraverso una tabella ma è assorbita indifferentemente da una delle entità legate.

Analizzando il nostro caso ci troviamo di fronte ad un modello piuttosto chiaro e senza ambiguità per cui la maggior parte di queste problematiche non capitano.

Fatte queste considerazioni, è ora di passare dal modello concettuale presentato al modello fisico, operazione detta di "traduzione" del modello.

Ci siamo avvalsi per questa operazione dello strumento CASE **PowerDesigner**, specificando che il DBMS che verrà utilizzato è MySql.

Nell'appendice B è riportato il risultato di tale operazione, insieme al processo per il caricamento iniziale dei dati.

Prima di effettuare la traduzione notiamo come nel nostro schema concettuale non ci sia la presenza di attributi multivalore o generalizzazioni, che ci imporrebbero delle strategie per risolverle e gestirle.

Possiamo quindi procedere direttamente alla costruzione del modello fisico.

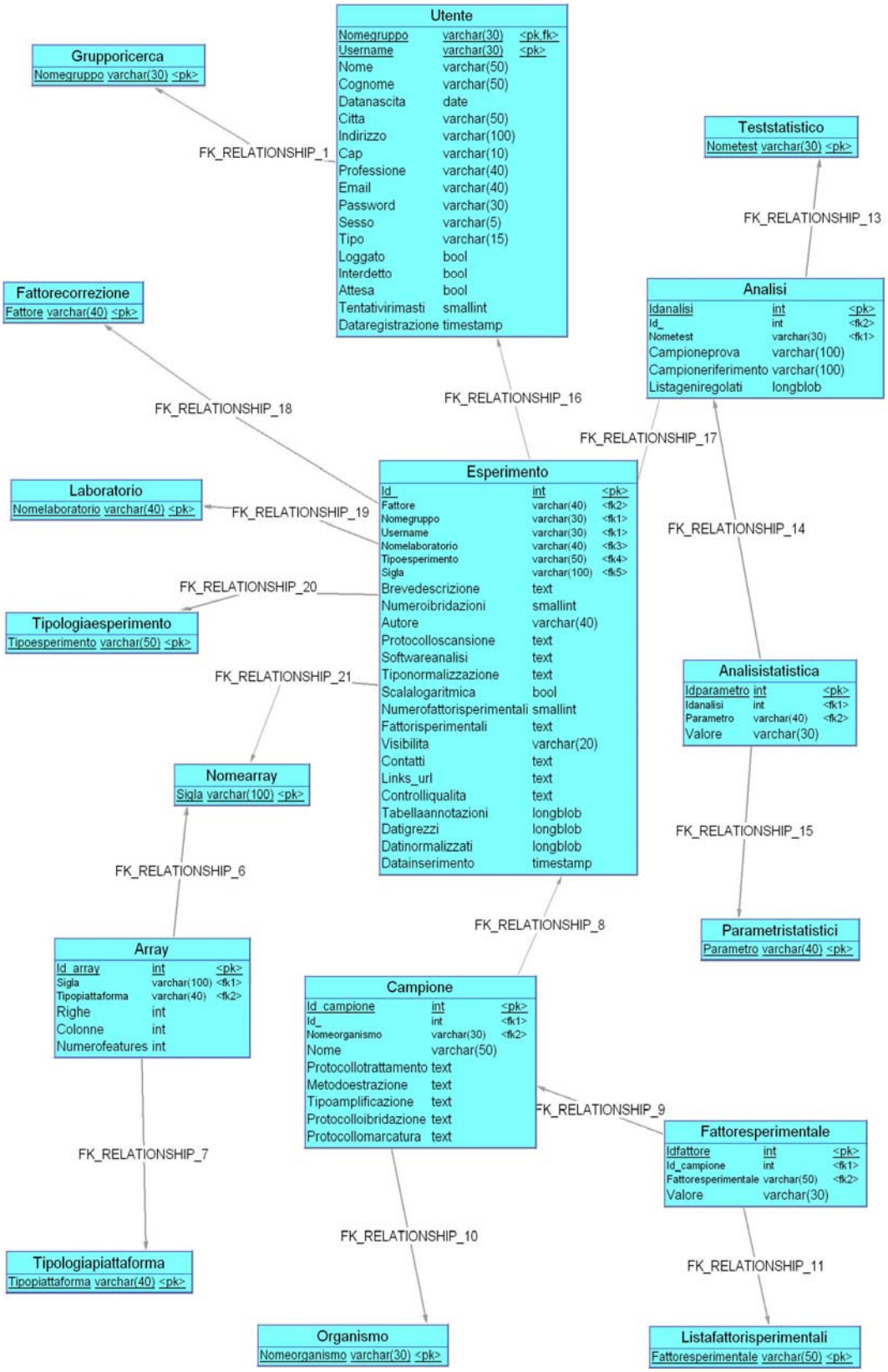


Figura 4.22 – ***Physical Model***

Sulla base delle regole suddette, il modello fisico risultante ha le seguenti caratteristiche:

Nella maggior parte delle tabelle è stato introdotto un attributo (colonna) che ne rappresenti la chiave primaria, con un nome del tipo "idNomeTabella" per esprimere che si tratta di un identificativo univoco;

Le relazioni uno-a-molti tra gruppo di ricerca e utente, tra utente e esperimento, tra esperimento e analisi, tra esperimento e campione, tra esperimento e array, tra campione e fattoresperimentale sono state eliminate aggiungendo la chiave della relazione presente dal lato "uno" come attributo della relazione dal lato "molti".

Oltre a queste trasformazioni sono stati introdotti dei particolari vincoli (constraints) di "*cascade*" sugli eventi "*onDelete*" , in modo da non lasciare informazioni inutili che rendano il database "sporco".

In particolare il vincolo di cascade sull'evento onDelete è stato posto tra le seguenti relazioni:

- Tra "gruppo di ricerca" e "utente", così alla cancellazione di un gruppo di ricerca corrisponderà l'eliminazione di tutti gli utenti facenti parte di quest'ultimo.
- tra "esperimento" e "campione", così alla cancellazione di un esperimento corrisponderà l'eliminazione di tutti i campioni facenti parte di quest'ultimo.
- tra "campione" e "fattore sperimentale", così alla cancellazione di un campione corrisponderà l'eliminazione di tutti i fattori sperimentali analizzati su quest'ultimo.
- tra "esperimento" e "analisi", così alla cancellazione di un esperimento corrisponderà l'eliminazione di tutte le analisi fatte sullo stesso.
- tra "analisi" e "analisistatistica", così alla cancellazione di un' analisi corrisponderà l'eliminazione di tutti le coppie parametro statistico-valore corrispondenti a quest'ultimo.

- tra "fattore sperimentale" e "campione", così alla cancellazione di un fattore sperimentale corrisponderà l'eliminazione di tutti i campioni analizzati con quel fattore.
- tra "organismo" e "campione", così alla cancellazione di un organismo corrisponderà l'eliminazione di tutti i campioni che presentano quell'organismo.

In particolare, per tutti i casi descritti prima, è stato esteso a cascade anche l'evento *"onUpdate"*, per far sì che le modifiche si propaghino correttamente a tutte le tabelle legate e che il database non risulti "sporco" in seguito ad operazioni di aggiornamento e modifica dei dati.

CAPITOLO CINQUE

IMPLEMENTAZIONE DELL'APPLICAZIONE MICROARRAY WEB

5.1 Perché un'applicazione web e non un sito web?
5.2 Analisi delle tecnologie utilizzate nella realizzazione dell'applicazione
5.3 L'applicazione Microarray Web
5.4 Screenshots dell'applicazione
5.5 Internazionalizzazione (I18N)
5.6 Validazione
5.7 Connessioni al database e "teoria del pooling"
5.8 Inserimento sicuro dei dati e uso delle "transazioni".
5.9 Il framework Tiles.
5.10 L'implementazione Myfaces
5.11 L'applet Vimida
5.12 Commons Email

Questo capitolo illustra la fase di implementazione dell'applicazione web realizzata. Lo scopo di quello che diremo nelle prossime pagine è capire perché durante il processo di sviluppo sono state prese determinate decisioni rispetto a possibili alternative e illustrare le principali peculiarità dell'applicazione in esame.
Nel fare ciò abbiamo cercato, laddove fosse necessario, di introdurre fondamenti teorici utili alla migliore comprensione delle scelte fatte.

5.1 Perché un'applicazione web e non un sito web?

Partiamo immediatamente dalla risposta, o meglio da 2 definizioni:

- *Sito Web:* software che mostra ad un utente informazioni in una modalità di "sola lettura", senza cioè che esso possa interagire per modificare lo stato del software stesso ed in particolare i dati;
- *Applicazione Web:* software utilizzante la stessa infrastruttura di un sito, che consente ad un utente di interagire con esso, permettendo la modifica dello stato del software.

Un sito Web è un modo per diffondere documenti e più in generale informazioni sulla rete internet. Per un utente è possibile, attraverso la rete, visualizzare e consultare pagine di testo, oppure contenuti multimediali di varia natura e di varia utilità.

Un sito Web è in sostanza un deposito remoto di informazioni disponibili più o meno a tutti.

Più complesse dei siti Web sono le applicazioni Web, che rappresentano un modo per realizzare un'interazione utente-dati remoti: l'utente è attivo ed ha la possibilità di cambiare il contenuto di informazioni o di interagire con esse.

Tutte le applicazioni Web svolgono essenzialmente lo stesso compito:

1) forniscono un'interfaccia all'utente per l'immissione di dati, che costituiscono una richiesta verso un servizio remoto

2) trasmettono queste richieste attraverso la rete

3) processano queste richieste utilizzando dati remoti per processare queste richieste

4) trasmettono all'utente i relativi risultati

5) visualizzano all'utente i risultati delle richieste

Considerando la seguente figura, si può delineare un tipico flusso di operazioni (workflow) che coinvolge il funzionamento di un'applicazione Web:

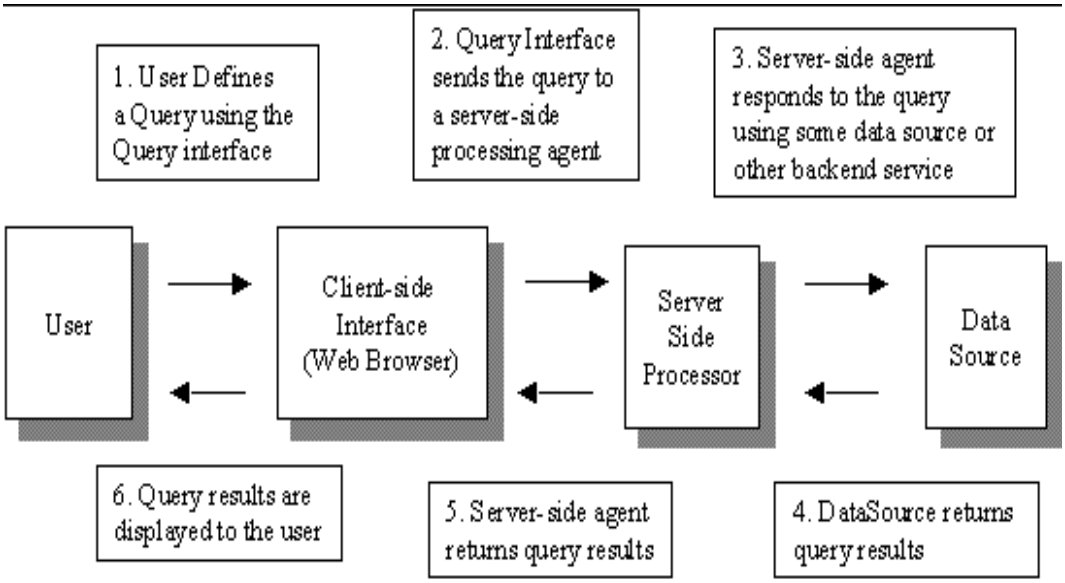


Figura 5.1 – ***Funzionamento di un'applicazione web***

1) un utente effettua una richiesta, magari riempiendo dei campi dell'interfaccia utente;

2) la richiesta con i relativi dati viene inoltrata attraverso la rete al server

3) il server processa la richiesta utilizzando dei dati oppure altri servizi

4) il server raccoglie il risultato della richiesta

5) il risultato viene inoltrato all'utente

6) all'utente viene visualizzato il risultato.

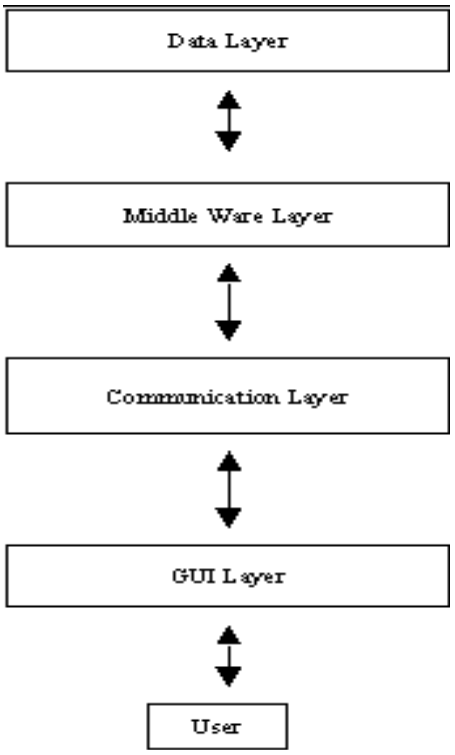
5.2 Analisi delle tecnologie utilizzate nella realizzazione dell'applicazione.

Analizzando il flusso di realizzazione di un'applicazione web, ciò che emerge è che possono essere utilizzate diverse tecnologie e approcci in ogni fase.

In pratica, vengono utilizzati molteplici strumenti e meccanismi che, combinati in vario modo, realizzano questo workflow. Questi meccanismi possono essere raggruppati in quattro diverse catagorie concettuali.

Rappresentando il tutto come un insieme di strati, ogni tecnologia usata appartiene ad uno dei seguenti strati (o layer):

a. lo strato dei dati
b. lo strato intermedio o middle ware
c. lo strato della comunicazione
d. lo strato dell'interfaccia utente



Analizziamo più nel dettaglio questa classificazione.

GUI layer

Lo strato dell'interfaccia utente, detto GUI (Graphical User Interface) layer, è lo strato vicino all'utente, pertanto il suo ruolo principale è di fare da intermediario tra il mondo umano e quello del computer.

In generale, ha il compito di tradurre le richieste dell'utente in un linguaggio comprensibile al computer e di tradurre i dati forniti dai risultati delle richieste in un formato comprensibile all'utente. In particolare ha anche altri compiti importanti, come validare gli input dell'utente, gestire errori, fornire una guida d'utilizzo ecc…

In definitiva deve rendere semplice l'utilizzo dell'applicazione da parte dell'utente, colmando il famoso "gap semantico" tra l'uomo e la macchina.

Questa interfaccia è tipicamente realizzata da un programma del lato client detto web browser. Il web browser è in grado di contattare un server, richiedergli un documento e visualizzare il documento fornito dal server all'utente. I dettagli di ciò fanno parte del communication layer, ma per quanto riguarda il GUI layer va considerato che , per la visualizzazione corretta di documenti e per altre funzionalità di interazione con l'utente, vi sono nel browser vari meccanismi utilizzabili.

Innanzitutto un web browser è in grado di interpretare un linguaggio di formattazione per la visualizzazione di informazioni, tipicamente HTML.

Html

HTML è un linguaggio ipertestuale, cioè, oltre a realizzare la formattazione delle informazioni, permette all'utente di scegliere le informazioni da visualizzare mediante collegamenti ipertestuali.

Ogni file HTML è un file di testo in cui vi sono le informazioni da visualizzare e delle direttive per la visualizzazione.

Attualmente i web browser hanno aumentato le loro funzionalità, divenendo degli interpreti, capaci di elaborazioni anche complesse. Questa evoluzione, dovuta alle esigenze dell'utente ed alle più ampie risorse dei computer, ha portato allo sviluppo di varie tecnologie nel GUI layer.

Il web browser è in grado di interpretare vari linguaggi che estendono le funzionalità dell' HTML, oppure di eseguire componenti già compilati che vengono dalla rete.

Come linguaggi che estendono HTML vi sono Javascript, VBScript ecc…

Per i componenti precompilati si utilizzano linguaggi come il Java, per la sua portabilità.

Css

Dietro il semplice acronimo CSS (Cascading Style Sheets - Fogli di stile a cascata) si nasconde uno dei fondamentali linguaggi standard del W3C. La sua storia cammina su binari paralleli rispetto a quelli di HTML, di cui vuole essere l'ideale complemento.

Da sempre infatti, nelle intenzioni degli uomini del Consortium, HTML, così come la sua recente evoluzione, XHTML, dovrebbe essere visto semplicemente come un linguaggio strutturale, alieno da qualunque scopo attinente la presentazione di un documento. Per questo obiettivo, ovvero arricchire l'aspetto visuale ed estetico di una pagina, lo strumento designato sono appunto i CSS.

L'ideale perseguito da anni si può sintetizzare con una nota espressione: *separare il contenuto dalla presentazione.*

La prima specifica ufficiale di CSS (**CSS1**) risale al dicembre del 1996.

Quasi tutto quello che con HTML non si riesce a realizzare, sarà possibile ottenerlo con i fogli di stile.

Grazie ai Css è ora possibile, infatti, dare al testo delle nostre pagine un aspetto da word-processor: non solo con il colore o i font che preferiamo, ma con un sistema di interlinea pratico e funzionale, con le decorazioni che desideriamo, riuscendo a spaziare lettere e parole, impostando *stili diversi* per titoli e paragrafi, sfruttando i benefici dell'indentatura o della giustificazione e tanti altri vantaggi.

Nella nostra applicazione abbiamo fatto largo uso dei Css, personalizzando ogni aspetto delle pagine web. La cosa più importante è che, grazie all'introduzione dei CSS, la gestione della nostra applicazione web, o più in generale di un sito, è notevolmente semplificata.

Modificare lo stemma del microarray presente in tutte le pagine non comporta la modifica di tutte le pagine, perché i CSS sono separati dal documento. Basta aprire il foglio di stile, cambiare l'immagine e tutte le pagine mostreranno le nuove modifiche.

Il risultato sono pagine più leggere e facili da modificare.

Communication layer

Lo strato della comunicazione (communication layer) ha il compito di trasmettere le informazioni immesse dall'utente al server. Questo strato presenta una grande complessità per la varietà di scenari e problemi possibili. In generale, in questo strato, sono implementati i protocolli di comunicazione, che permettono a tutti i partecipanti alla comunicazione di capirsi. Il protocollo usato sulla rete è il TCP/IP su cui si appoggia il protocollo HTTP, realizzando una struttura a livelli di protocolli che richiama la pila ISO-OSI.

Middle layer

Lo strato intermedio (middlelayer) ha il compito di accettare le richieste provenienti dallo strato della comunicazione e di elaborarle. Le richieste vengono evase o utilizzando risorse locali al Web server o risorse distribuite sulla rete. Per realizzare ciò vi sono varie alternative, tra cui la creazione di servizi sulla macchina server in grado di rispondere alle richieste che provengono dalla rete, di elaborarle e di fornire una risposta.

Il modo più usato per implementare questi servizi è attraverso il linguaggio Java, col quale o si realizzano direttamente servizi Java oppure si realizzano le cosiddette servlets Java.

Data layer

Lo strato dei dati ha il compito di gestire e contenere i dati di interesse per l'utente. Vi sono vari modi di gestire i dati, ma il più vantaggioso ed il più diffuso è attraverso un database. Il database può risiedere sul server che gestisce le richieste dalla rete o può risiedere su un'altra macchina. Il primo caso rientra nella architettura client-server semplice, mentre il secondo caso rappresenta un'architetture datte 'three-tier', in cui i tre attori di una transazione su Web, cioè client, Web server e database server, sono sostenuti da macchine diverse.

Il layer più importante in un'applicazione Web è il middleware layer, dove si realizza la maggior parte della logica applicativa e dove avvengono le scelte implementative più critiche: negli altri strati le scelte sono più o meno vincolate e obbligate. La prima scelta importante nell' applicazione MicroarrayWeb è stata di utilizzare il linguaggio Java per la realizzazione del middleware layer..

Questo, in pratica, significa riferirsi all'ambiente J2EE.

Il J2EE (Java 2 Platform Standard Edition) è un ambiente per lo sviluppo, la costruzione, l'implementazione di applicazioni Web basate sul linguaggio Java ed indipendenti dalla piattaforma sulla quale operare.

Esso prevede un'insieme di servizi, API , protocolli per la realizzazione di applicazioni Web multi-strato (multitiered).

Http

Se da un lato HTML (Hypertext Markup Language) è il linguaggio utilizzato per descrive gli elementi dei documenti Web, HTTP (Hypertext Transfer Protocol) è il "linguaggio" utilizzato per controllare l'invio di questi documenti via Internet.

Le specifiche del protocollo sono attualmente in carica al W3C (World Wide Web Consortium).

Il protocollo HTTP prescrive le regole mediante le quali i browser effettuano le richieste e i server forniscono le relative risposte. Queste regole includono i metodi per:

- richiedere un documento per nome
- accordarsi sul formato dei dati
- determinare l'identità dell'utente
- decidere come gestire le risorse non aggiornate
- indicare i risultati di una richiesta

e altre funzioni utili.

Il protocollo HTTP è costituito da un insieme di comandi scritti come normale testo ASCII. Quando si usa il browser Web non si introducono direttamente i comandi http; al contrario, quando si digita un indirizzo URL o si fa clic su un collegamento ipertestuale, il browser traduce questa azione nei corrispondenti comandi HTTP che richiedono il documento dal server specificato nell'indirizzo URL. Il server Web trova il documento e lo invia al browser, dove viene visualizzato insieme alle immagini e ai collegamenti ipertestuali a esso associati.

Gli standard internet vengono normalmente specificati in un documento RFC (Request for comments) pubblicato dalla IETF (Internet Engineering Task Force).

I documenti RFC sono numerati e dopo la pubblicazione non cambiano mai. Ogni volta che viene aggiornato uno standard, viene emesso un nuovo documento RFC. Vi sono vari documenti RFC che riguardano il protocollo http, l'ultimo attualmente è l'RFC 2616 che è una versione aggiornata delle specifiche 1.1.

Le specifiche descrivono http come un protocollo senza stati a richieste e risposte le cui operazioni di base sono le seguenti:

1. Un'applicazione client, ad esempio un browser Web, apre un socket verso la porta http del server Web (normalmente la porta 80)
2. Attraverso questa connessione il client scrive una riga di richiesta di testo ASCII seguita da zero o più intestazioni http, da una riga vuota e dai dati che accompagnano la richiesta.
3. Il server Web analizza la richiesta e individua la risorsa specificata.
4. Il server scrive una copia della risorsa sul socket, dove può essere letta dal client.
5. Il server chiude la connessione.

E' fondamentale considerare il fatto che questo modello è *senza stati*.

Questo significa che il server Web non ricorda nulla delle precedenti richieste pervenute dallo stesso client. Il protocollo considera semplicemente la richiesta di un documento e la risposta costituita dal documento stesso. Questo impone importanti limiti nella programmazione delle applicazioni.

Per risolvere questa situazione, il server deve assegnare un identificatore alla sessione e deve fare in modo che il client ricordi l'identificatore e lo fornisca al server a ogni richiesta.

E' quanto faremo vedere in seguito, quando nella nostra applicazione introdurremo il concetto di sessione per memorizzare lo stato dell'utente durante la navigazione.

Per quel che riguarda il protocollo, attualmente è molto diffusa la versione 1.0, ma è anche disponibile la versione 1.1 che, comunque, è in corso di valutazione. Il protocollo HTTP 1.0 viene spiegato nella RFC 1945.

Tornando ai messaggi che si scambiano client e server nel protocollo, il messaggio di richiesta è composto di tre parti: linea di richiesta (request line), sezione header (informazioni aggiuntive), Body (corpo del messaggio).

La linea di richiesta è composta dal metodo, URI e versione del protocollo.

L'URI sta per Uniform Resource Identifier ed indica l'oggetto della richiesta (ad esempio la pagina web che si intende ottenere).

I metodi HTTP più comuni sono GET e POST. Utilizzando il metodo GET, i dati vengono appesi alla request-URI nella forma di coppie *chiave=valore* separati tra di loro dal carattere "&". La stringa seguente è un esempio di campo request-URI per una richiesta di tipo GET.

Il metodo POST pur producendo gli stessi effetti del precedente, utilizza una modalità di trasmissione dei dati differente impacchettando i contenuti dei campi di un form all'interno di un message-header. Rispetto al precedente, il metodo POST consente di inviare una quantità maggiore di dati ed è quindi utilizzato quando è necessario inviare al server nuovi dati affinché possano essere memorizzati all'interno della Base Dati.

Il messaggio di risposta è composto dalle seguenti tre parti: linea di stato (status line), sezione header, body (contenuto della risposta).

La linea di stato riporta un codice a tre cifre catalogato nel seguente modo:

- **2xx:** Success
- **3xx:** Redirection
- **4xx:** Client error
- **5xx:** Server error

Nel caso più comune il server risponde con un codice 200 (OK) e fornisce contenuto nella sezione body. Altri casi sicuramente incontrati da tutti sono:

- **302 Found**. La risorsa è raggiungibile con un altro URI indicato nel header Location. Di norma i browser eseguono la richiesta all'URI indicato in modo automatico senza interazione dell'utente.
- **404 Not Found**. La risorsa richiesta non è stata trovata e non se ne conosce l'ubicazione. Di solito avviene quando l'URI indicato in modo incorretto od è stato rimosso il contenuto dal server.
- **500 Internal Server Error**. Il server non è in grado di rispondere alla richiesta per un suo problema interno.

Gli header della risposta più comuni sono:

- **Server**. Indica il tipo la marca , la versione del server.
- **Content-Type**. Indica il tipo di contenuto restituito. La codifica di tali tipi

  (detti Media type) sono registrati presso IANA (Internet Assigned Number Authority) e sono detti tipi MIME (Multimedia Internet Message Extensions), la loro codifica è descritta nel documento RFC1521. I principali sono:

  text/html. Documento HTML
  text/plaintext. Documento di testo non formattato
  image/jpeg. Immagine di formato Jpeg

Dal momento che tutto il traffico HTTP è anonimo e *in chiaro*, sono state sviluppate diverse alternative per garantire differenti livelli di sicurezza. Ad oggi quello più usato è il protocollo HTTPS. Il meccanismo HTTPS, inventato da Netscape usa il sottostante canale cifrato a livello di trasporto mediante SSL o TLS per impedire l'intercettazione di qualsiasi parte della transazione. In pratica è un normale http inglobato in un Secure Sockets Layer (SSL), un sistema di comunicazione che garantisce la privacy quando si comunica con altre applicazioni abilitate all'SSL.

Una connessione SSL può essere stabilita tra un client e un server solo quando entrambi i sistemi sono in esecuzione in modalità SSL e hanno la capacità di autenticarsi reciprocamente.

Il Java

Il linguaggio Java fu creato per l'utilizzo in piccoli dispositivi embedded e questo lo ha fortemente caratterizzato.

La creazione di Java è partita da un progetto della SUN Mycrosystems per realizzare software per dispositivi di controllo di elettrodomestici.

Questo contesto vincolò fortemente il linguaggio poiché questi dispositivi hanno poca memoria e scarsa potenza. Inoltre era fondamentale la portabilità dei programmi scritti in Java per l'indipendenza dalla piattaforma hardware sulla quale operare. Si realizza ciò compilando il codice Java in un formato intermedio detto 'Bytecode' che può essere eseguito su qualsiasi macchina, mediante un macchina virtuale detta JVM.

Grazie a queste sue peculiarità ed al fatto di essere basato sul paradigma ad oggetti il Java si è diffuso oltre le iniziali aspettative. Il Java cominciò ad essere usato nel Web per realizzare elaborazioni e contenuti dal lato client attraverso le applets.

Ma dopo pochi anni si diffuse il suo utilizzo come piattaforma lato server. Attualmente è considerato il linguaggio ideale per elaborazioni sul server.

Il punto debole del Java è la sua efficienza: essendo il suo codice eseguito non direttamente sulla macchina, ma eseguito da una macchina virtuale che assorbe risorse, la sua efficienza nell'utilizzo di risorse è più bassa di un programma che effettuasse le stesse elaborazioni, ma compilato in codice macchina.

Mysql

"MySQL is a very fast, multi-threaded, multi-user and robust SQL (Structured Query Language) database server" (MySQL è un database SQL server molto veloce, multi-processo, multi-utente e robusto). Questa è la definizione di MySQL datane dal team di sviluppo nella documentazione. Analizziamo in dettaglio i termini che la compongono:

1. Database: è un archivio per gestire dati, strutturato in modo semplice e regolare. I dati sono organizzati in tabelle. Ogni tabella è organizzata in righe e colonne. Le righe possono contenere diverse parti di informazione.
2. SQL (Structured Query Language): è un linguaggio standard che permette di immagazzinare, aggiornare e accedere alle informazioni contenute in un database, di qualunque tipo esse siano (testo, immagini, file in generale). MySQL è basato su SQL.
3. Multi processo: significa che può rispondere a diverse richieste contemporaneamente.
4. Multi utente: significa che può essere utilizzato contemporaneamente da diversi utenti.
5. Robusto e veloce: significa che è in grado di gestire grandi quantità di dati con risorse hardware limitate. MySQL è stato sviluppato per poter gestire database con centinaia di migliaia di tabelle: il suo team di sviluppo lo sta utilizzando dal 1996 su un database con 10,000 tabelle, delle quali più di 500 hanno più di 7 milioni di record, per circa 100 (cento!) gigabyte di spazio.

E' da tener presente che uno dei punti di forza di MySQL è la sua universalità di applicazione. Si adatta infatti in modo ottimale sia ad ambienti Windows, sia ad ambienti Unix/Linux.

E' facile da installare, è facile da eseguire come servizio ed è facile connettersi.

L'aumento esponenziale della diffusione di MySQL è spiegabile con la diffusione di Linux e dell'enorme successo del movimento Open Source: risultato dell'iniziativa di molti produttori di software che hanno deciso di fornire sia il prodotto, sia il codice sorgente, consentendo all'utente di vedere come funzionano i programmi e di modificarli secondo le proprie esigenze.

Compatibilmente con la filosofia Open Source, la distribuzione di MySQL, se per uso personale, è gratuita, il suo codice sorgente è visibile al pubblico e modificabile.

Le performance di MySQL sono stupefacenti se paragonate ai suoi diretti concorrenti sul mercato,per averne un'idea è sufficiente consultare

http://www.mysql.com/benchmarks.html

Tomcat 5.5

Per quanto riguarda il Web Server, la scelta è caduta su **Apache Tomcat** realizzato nell'ambito del progetto Jakarta dalla fondazione Apache, fra i più sicuri e testati per questo genere di applicativi.

Il progetto *Apache Jakarta* è un progetto nato in seno alla Apache Software Foundation che offre un'insieme di soluzioni e librerie basate su Java. E' in realtà un contenitore di altri sotto-progetti quali Tomcat, Struts, ecc.

L'*Apache Software Foundation* è ,invece,una fondazione no-profit ed una comunità distribuita di sviluppatori che lavorano su progetti open-source. Ciascun progetto è gestito da un team di volontari che sono i contributori attivi al progetto.

In realtà Tomcat nasce come Application Server, molto spesso utilizzato in combinazione con il Web Server Apache (altre soluzioni spesso praticate sono Microsoft Internet Information Server, versione 4.0 o successiva, Microsoft Personal Web Server, versione 4.0 o successiva, Netscape Enterprise Server, versione 3.0 o successiva)

E' comunque possibile utilizzare Tomcat come Web Server; nel progetto in esame si è preferita questa soluzione perché Tomcat funge anche da Container per applicazioni web, conforme alle specifiche della piattaforma Java 2 Enterprise Edition(J2EE) .

Un *Container*, in generale, fornisce un ambiente che ospita componenti software che vanno in esecuzione all'interno di tale "contenitore". Esso rende disponibili dei servizi generali che i componenti all'interno di tale ambiente possono utilizzare. In particolare, Tomcat permette di effettuare il deploy e di eseguire al suo interno servlet, JSP e altre classi Java.

Volendo sottolineare una differenza semantica tra le definizioni di application server e web server potremmo dire che *un Application Server è un'applicazione che fornisce l'infrastruttura e le funzionalità di supporto a integrazione, deploy ed esecuzione di applicazioni e componenti server in un contesto distribuito.*

Dall'altro lato invece, *un Web Server è un programma che riceve le richieste http dal browser e restituisce la pagina da visualizzare tramite una risposta http.*

Quindi in base alle esigenze di utilizzo si dovrà installare anche un web server con cui Tomcat coopererà: quando arriva una richiesta di esecuzione di una servlet o JSP, il web server la passa a Tomcat che si occupa di esaudirla.

Tecnologie JSP e Servlet

Per la scelta della tecnologia realizzativa sono state effettuate alcune considerazioni, al fine di individuare la metodologia più idonea alla realizzazione dell'applicazione.

Nel corso degli anni sono state sviluppate e standardizzate varie tecnologie per la realizzazione di pagine dinamiche, ciascuna con i propri pregi e difetti. Analizziamo le varie possibilità:

- *Common Gateway Interface* (*CGI*), è il primo e più semplice standard architetturale per la creazione di pagine dinamiche. Il principio di funzionamento è molto semplice, il Web Server riceve una richiesta http da un browser(che identifica uno script CGI), salva le informazioni interne alla richiesta in memoria, riconosce qual è lo script CGI richiesto e lo invoca.

  Lo script CGI può contenere interrogazioni a qualche base di dati oppure solo elaborazioni che portano alla produzione dinamica della pagina HTML che verrà inviato al Web Server, il quale a sua volta utilizzerà la pagina per costruire la risposta http da inviare al browser.

  A fronte del vantaggio di poter costruire pagine dinamicamente, ci sono alcune serie limitazioni, come ad esempio l'enorme esigenza di risorse. Infatti per ogni richiesta http viene creato un nuovo processo del Sistema Operativo che viene terminato dopo la costruzione della pagina dinamica.

  Questo continuo avvio e distruzione di processi produce un elevato sovraccarico delle prestazioni del sistema. Altro notevole limite è dovuto al fatto che la terminazione del processo CGI dopo l'evasione della richiesta non consente di avere a disposizione in memoria centrale strutture dati che possano essere condivise tra diverse richieste. Tali limitazioni hanno portato all'ideazione di architetture più potenti;

- *Servlet Java*, forniscono un metodo component-based e indipendente dalla piattaforma, per costruire web application. Esse creano un solo processo e consentono a ciascuna richiesta utente di utilizzare un thread più leggero, che viene gestito dalla Java Virtual Machine(JVM6). Ciascuna richiesta è associata con un thread separato, quindi thread o utenti multipli possono invocare lo stesso metodo contemporaneamente.

Un vantaggio significativo è dato dal fatto che essendo scritte in Java possono sfruttare l'intera e ricca serie delle Application Programming Interfaces (API) Java, tra cui JDBC ed EJB, e delle libreria Java. Le servlet non vengono direttamente eseguite da un Web server, ma hanno bisogno di un contenitore, servlet container, che le ospiti.

Il web server e il servlet container lavorano insieme per soddisfare le richieste. Esistono molti container disponibili, quindi le servlet non dipendono da un particolare produttore o da una specifica piattaforma.

A fronte di tutti questi vantaggi risultano anche degli svantaggi, dovuti principalmente al fatto che le servlet devono incorporare stabilmente output HTML.

Se, dopo aver sviluppato un sito di grandi dimensioni, si volesse cambiare l'estetica bisognerebbe rivedere l'intero codice e ricompilare la servlet. Un altro problema da non sottovalutare è legato alle responsabilità, perché i web designer costruiscono pagine HTML, ma non sempre sono a conoscenza di linguaggio Java, per cui questo mescolamento di Html e Java nelle servlet comporta dei tempi di sviluppo e verifica maggiori di altre soluzioni;

- *Java Server Pages*(*JSP*), rappresentano la naturale estensione delle Servlet Java, infatti dopo una preventiva elaborazione vengono trasformate in servlet. Esse sono composte da codice HTML, con all'interno scriptlet JSP e tag. Viene così risolto il problema dell'inglobamento del codice HTML nelle servlet, infatti può essere prima creata la grafica in HTML dai web designer e poi aggiunti gli scriptlet e i tag dagli sviluppatori JSP.

In particolare, per l'utilizzo della tecnologia JSP sono state ideate due architetture:

1. *Architettura JSP Model 1,* prevede che la pagina JSP gestisca tutta l'elaborazione della richiesta e ha la responsabilità di mostrare il risultato al Client, ovvero contengono al loro interno logica applicativa e logica di controllo del flusso;

2. *Architettura JSP Model 2*, prevede che la richiesta del Client venga prima intercettata da una servlet di controllo(o controller servlet) che gestisce l'elaborazione iniziale della richiesta e determina quale pagina JSP deve essere visualizzata come successiva. L'obiettivo è quello di avere nelle pagine JSP solo codice HTML e custom tag, con la minor presenza possibile di scriptlet, in quanto essi mischiano operazioni logiche e presentazione.

   La servlet di controllo è molto utile in quanto permette di effettuare controlli, quali ad esempio l'autenticazione e l'autorizzazione, prima di tutte le altre operazioni;

La differenza principale tra le due architetture sta nel fatto che nella architettura Model 2 c'è una chiara separazione tra business logic, presentation logic e control logic. Tale separazione viene spesso indicata come pattern *Model-View-Controller*, di cui abbiamo già ampiamente discusso nel capitolo relativo 3 relativamente al framework Jsf.

5.3 L'applicazione Microarray Web

5.3.1 Introduzione

L'applicazione Microarray Web nasce con lo scopo di aiutare gli addetti ai lavori nella memorizzazione, gestione e consultazione di dati di espressione genica ottenuti con la tecnica del microarray.

In particolare, data la complessità strutturale di un esperimento di questo tipo, si è cercato di realizzare procedure che descrivano in maniera quanto più dettagliata ed esaustiva possibile tutte le fasi dell'esperimento, favorendo all'utente l'inserimento dei dati attraverso semplici ed intuitive pagine web.

Il primo aspetto preso in considerazione riguarda la struttura dell'applicazione, in termini di eventuali moduli che la compongono, oltre che delle pagine, le immagini, gli eventuali fogli di stile (CSS) e di tutti gli altri elementi che ve ne fanno parte.

L'architettura del sistema prevede un unico modulo al quale è ovviamente associato un file di configurazione (*faces-config.xml*).

I file componenti il progetto riguardano le pagine ed i relativi contenuti, le immagini e il foglio di stile (*styles.css*) per la formattazione della grafica.

Infine, oltre alla librerie che contengono le classi costituenti l'architettura del framework, sono utilizzati i seguenti gruppi di ulteriori librerie :

Commons Mail : framework per l'invio delle mail basato su *JAF* (JavaBeans Activation Framework);

JDBC MySQL Connector : contiene le classi relative ai driver JDBC (Java DataBase Connectivity) per l'accesso alla base di dati realizzata con il DBMS MySQL;

Tiles Framework : framework per la formattazione delle pagine, con possibilità di stabilirne un formato fisso della struttura.

Un'ulteriore considerazione riguarda l'internazionalizzazione, supportata dalla Web application attraverso la definizione di due Resource Bundle.

In particolare, ne è previsto uno per la localizzazione di default, ossia l'Italia, e l'altro per un'ulteriore localizzazione supportata, in lingua Inglese.

Infine, ci sono da motivare le scelte relative alla realizzazione del Model; il sottolivello Data Access è completamente indipendente dagli altri.

Questo ne favorisce la portabilità, qualora si prenda in considerazione l'ipotesi di realizzare la Web application con un ulteriore framework differente.

Per quanto riguarda i sottolivelli di Business Logic ed External Interface, è stata fatta la seguente scelta: si è utilizzato l'approccio di "fusione", realizzando i backing beans in modo tale da contenere la business-logic e l'interfacciamento con le classi del framework.

Tale scelta ha il vantaggio di non dover realizzare un ulteriore strato di classi che implementi l'External Interface sublayer, il cui scopo sia quello di invocare i metodi sulle classi della business-logic ed utilizzare gli oggetti interni al framework. Lo svantaggio è ovviamente la perdita di portabilità, in quanto i backing beans utilizzano delle classi che non sarebbero presenti in framework diversi.

5.3.2 Controller

Il framework JSF fornisce le proprie classi di base che implementano la struttura del Controller di default, ma è possibile intervenire per poter personalizzare tali classi, in modo da aggiungere particolari funzionalità che vanno applicate durante la fase di comunicazione tra Model e View. Nel caso di studio in esame, si è reso necessario l'intervento sul Controller, per poter gestire due situazioni particolarmente delicate :

la gestione delle sessioni non concluse correttamente dell'utilizzatore;

la possibilità di accesso a pagine protette senza la necessaria operazione di login;

5.3.2.1 Gestione delle sessioni

Uno dei principali problemi di cui sono afflitte tutte le Web application è la gestione delle sessioni che non vengono concluse correttamente da un utente.

Nel caso specifico, quando l'utilizzatore, utente o amministratore che sia, effettua il login al sistema, viene memorizzato all'interno della base di dati l'informazione relativa all'accesso dell'utente, sintetizzata nel settaggio di un flag a true nella tabella degli utenti, che indica che l'utente è online, in modo che un tentativo fraudolento di accesso da parte di un secondo utente che disponga dei dati del primo venga ovviamente bloccato.

Inoltre viene controllato, sempre nella tabella degli utenti, la presenza di un valore >0 nel campo tentativi rimasti, che sta ad indicare che l'utente non è stato interdetto a causa di 3 errori consecutivi di digitazione della password.

Nell'effettuare correttamente la fase di logout, le informazioni di accesso vengono aggiornate settando opportunamente la variabile booleana "loggato" a false e ripristinando a 3 (il massimo consentito) il numero di tentativi consentiti all'utente per l'inserimento corretto della password.

Ci sono moltissime situazioni in cui un utente chiude direttamente il browser, senza completare correttamente il logout. In questo caso è necessario intervenire per ristabilire la coerenza delle informazioni contenute nel database rispetto alla realtà. L'evento di chiusura del browser non può essere intercettato dal server, sul quale è in esecuzione la Web application, ma ovviamente solo dal client.

Il linguaggio di scripting lato client, che permette di eseguire elaborazioni su quest'ultimo è ovviamente il JavaScript, il quale però non mette a disposizione la possibilità di intercettare un semplice evento come la chiusura di una finestra del browser, a meno che questa azione non venga eseguita sempre mediante codice.

Per questo ed altri motivi, è stato introdotto il concetto di sessione, mediante la quale si stabilisce un legame tra l'utente che utilizza l'applicazione ed il server su cui quest'ultima è in esecuzione.

Alla sessione è associato un concetto di timeout, per cui se non si rileva attività da parte dell'utente fino allo scattare di quest'ultimo, ne viene eseguita l'invalidazione.

E' in corrispondenza di questo evento che si può intervenire effettuando le operazioni che garantiscano la coerenza delle informazioni nel sistema.

Per risolvere questa problematica ci siamo rivolti al package javax.servlet.http che costituisce l'API per le sessioni. Esso contiente 3 classi:

HttpSession: un'interfaccia che si comporta come una mappa o come una tabella hash e che è in grado di memorizzare e prelevare gli oggetti per nome.

Viene creata una sessione tramite una chiamata a HttpServletRequest.getSession() e questa sessione dura per un determinato periodo o finchè non viene esplicitamente chiusa.

Le richieste http in arrivo che trasportano l'identificatore di sessione vengono associate automaticamente alla sessione.

HttpSessionBindingListener: un'interfaccia che consente ad un oggetto di conoscere quando è stato memorizzato o rimosso da una sessione. L'interfaccia contiene i metodi callback *valueBound()* e *valueUnbound()* , che l'oggetto deve implementare per poter ricevere le notifiche di associazione.

HttpSessionBindingEvent: un oggetto passato ai metodi *valueBound()* e v*alueUnbound()* di HttpSessionBindingListener. L'evento offre dei metodi per restituire la sessione e il nome col quale l'oggetto è stato connesso alla sessione.

Nella nostra applicazione abbiamo deciso di memorizzare i dati di accesso dell'utente e il locale in uso in un oggetto Visit, memorizzato nella sessione.

Per ottenere ciò abbiamo richiamato nel metodo di autenticazione, nella pagina di login

```
HttpSession session = (HttpSession)
facesContext.getExternalContext().getSession(true);
Visit visit=new Visit();
visit.setAuthenticationBean(this);
setVisit(visit);
session.setAttribute("visit",visit);
```

creando così una sessione, un oggetto Visit, memorizzando le informazioni in Visit e registrando Visit nella sessione.

Per poter gestire la chiusura anticipata e imprevista del browser abbiamo optato per far implementare alla classe Visit l'interfaccia HttpSessionBindingListener

```
public class Visit implements HttpSessionBindingListener {
public void valueBound(HttpSessionBindingEvent event) {
        try{
                DataSource dataSource = Singleton.getInstance().getDataSource();
                Connection conn = dataSource.getConnection();

                Statement stmt=null;
                stmt = conn.createStatement();
                String query = "UPDATE utente set loggato='1'"
                                + "WHERE username = '"
                                + authenticationBean.getLoginName()
                                + "' "
                                +                "AND                nomegruppo
='authenticationBean.getTeamName()
                                + "'";
                int Nrows2 = stmt.executeUpdate(query);

        }
        catch(Exception e){
                e.printStackTrace();
                        }       }
public void valueUnbound(HttpSessionBindingEvent event) {

        try{
                DataSource dataSource = Singleton.getInstance().getDataSource();
                Connection conn = dataSource.getConnection();
```

```
                Statement stmt=null;
                stmt = conn.createStatement();
                String query = "UPDATE utente set loggato='0'"
                                + "WHERE username = '"
                                + authenticationBean.getLoginName()
                                + "'AND nomegruppo ='" +
                               authenticationBean.getTeamName() + "'";
                int Nrows = stmt.executeUpdate(query);

        }
        catch(Exception e){
                e.printStackTrace();

        }


}
```

In questa maniera leghiamo all'oggetto Visit (che rappresenta la sessione dell'utente) i due metodi di valueBound() e valueUnbound().

All'atto del

```
session.setAttribute("visit",visit);
```

viene richiamato il metodo valueBound().

Inoltre, dichiarando nel web.xml dell'applicazione un timeout per la sessione(in minuti)

```
<!--  time out di sessione utente -->
 <session-config>
  <session-timeout>30</session-timeout>
 </session-config>
```

risolviamo il nostro problema, in quanto allo scadere del timeout la sessione è chiusa e il metodo *valueUnbound* è richiamato con il compito di ripristinare la coerenza della base di dati. La scadenza del timeout indica che l'utente non sta più generando attività sul server e presumibilmente ha chiuso il browser senza eseguire correttamente la procedura di Logout.

5.3.2.2 Protezione delle pagine

La seconda problematica in cui è necessario l'intervento sul Controller riguarda la possibilità che un utilizzatore voglia accedere alle pagine protette dell'applicazione senza avere eseguito correttamente l'operazione di login.

Questo è importante in un'applicazione come la nostra, dove i dati in questione hanno importanza notevole per il loro significato e per i costi impiegati per ottenerli.

Ovviamente, il sistema deve intercettare un'azione di questo tipo, redirezionando l'utente alla homepage, nella quale è costretto ad eseguire l'accesso.

Dall'analisi del problema, si evince che l'intervento deve essere eseguito per ciascuna richiesta che arrivi dal client, in modo da sincerarsi che l'utilizzatore sia correttamente loggato, qualunque sia la pagina a cui vuole accedere, escluse le pagine relative alla registrazione ed alla richiesta dei dati di accesso via mail.

Per la soluzione di questo secondo problema ci siamo affidati al concetto di Filtro.

I filtri sono una particolarità delle Servlet API e permettono di eseguire controlli addizionali su una richiesta prima e dopo che essa sia gestita dalle servlets tradizionali. Poiché Jsf è implementata come una servlet, è possibile usare filtri per eseguire compiti prima o dopo che Jsf processi le richieste.

Le API per i filtri sono definiti dalle interfacce Filter, FilterChain e FilterConfig nel package javax.servlet. Il metodo più importante in questa interfaccia è il doFilter, al quale sono passati la request, la response e l'oggetto Filter chain.

Normalmente i filtri sono usati per scopi di "autorizzazione" perché possono reindirizzare una richiesta se l'utente non è autorizzato a vedere la risorsa richiesta.

Questo è esattamente ciò che l'AuthorizationFilter realizzato fa.

In particolare recupera la sessione in corso e controlla se esista un oggetto Visit memorizzato.

Se esiste concede l'accesso, se non esiste redireziona l'utente alla pagina di Login.

Ciò avviene principalmente nel metodo doFilter()

```
if (visit == null) {
        session.setAttribute("originalTreeId", httpRequest.getPathInfo());
                Utils.log(servletContext, "redirecting to "+httpRequest.getContextPath() +
        "/faces/Login.jsp");
        httpResponse.sendRedirect(httpRequest.getContextPath()+ "/index.jsp");
}
else {
        chain.doFilter(request, response);
}
```

La cosa che si nota è che nel metodo doFilter() non esistono riferimenti alle API Jsf. Il metodo doFilter cerca un oggetto Visit usando l'oggetto standard delle servlet HttpSession, piuttosto che il ValueBinding di Jsf. Questo non deve meravigliarci, perché un filtro va in esecuzione prima che Jsf processi la request, per cui non ci sarebbe nessun istanza del FacesContext disponibile. Poiché il metodo Application.getValueBinding richiede un FacesContext non può essere usato nei filtri.

5.3.3 View

Per la realizzazione dell'interfaccia utente (UI) sono state utilizzate due librerie proprie del framework, ossia Core ed HTML, la prima per quanto riguarda aspetti avanzati come l'uso dei validatori, mentre la seconda per la costruzione dei form, la visualizzazione dei messaggi e la formattazione del contenuto delle pagine.

Le librerie standard

Per quanto riguarda le librerie di tag standard, il framework JSF utilizza solo ed esclusivamente le proprie librerie, Core ed HTML, in quanto non prevede alcun tipo di modifica nel Deployment Descriptor *web.xml*

I forms: contenuto e layout

Per la definizione della struttura dei form ed il posizionamento degli elementi all'interno della pagina, viene introdotto il concetto di Panel per la formattazione dei contenuti. Tale concetto è ben noto nell'ambito dello sviluppo delle applicazione desktop tramite le librerie AWT e Swing ed è stato introdotto nelle Web application, come passo verso un'unificazione tra le due tipologie di applicazioni.

La libreria JSF HTML fornisce i tag <*h:panelGrid*> e <*h:panelGroup*> di cui il primo definisce appunto un Panel, costituito da un fissato numero di colonne, all'interno del quale vengono disposti gli elementi mentre il secondo permette di raggruppare più campi, facendo in modo che nel Panel vengano trattati come un unico elemento.

Inoltre, all'interno del Panel è possibile specificare le intestazioni delle colonne, mediante il tag <*f:facet*>, così come gli stili da applicare, definiti all'interno di un file CSS esterno.

```
<h:panelGrid columns="4" rowClasses="oddRows,evenRows" bgcolor="#eee">

<h:outputText value="#{msg.FirstNameInput}" styleClass="outputText"/>
<h:inputText value="#{user.firstName}" id="FirstNameInput" required="true"
        styleClass="inputText">
</h:inputText>
<h:outputText value="#{msg.Required}"/>
<h:message for="FirstNameInput" styleClass="errors"/>

……..
<h:commandButton  value="#{msg.submitUser}" action="#{user.AddNewUser}"
```

```
            onclick="return confirm('Confermi la registrazione?');"/>
<h:messages styleClass="errors"/>
<h:commandButton value="#{msg.undo}" type="reset"/>
</h:panelGrid>
```

Costruzione condizionale dei contenuti

Un' ulteriore considerazione riguarda la costruzione condizionale di una pagina, che consiste nel fare in modo che un certo contenuto sia visualizzato o meno a seconda del verificarsi di opportune condizioni. Tale operazione viene eseguita mediante l'utilizzo dell'attributo *rendered* di cui è dotato ciascun tag della libreria JSF HTML.

Tale attributo può assumere un valore booleano che indica appunto se il tag ed il relativo contenuto debba essere renderizzato o meno. Facendo uso dell'Expression Language e del value-binding, è possibile definire un'espressione nella quale entrino in gioco le proprietà dei backing beans, in modo da esprimere condizioni complesse sulla base delle quali visualizzare o meno il componente.

Nella nostra applicazione abbiamo fatto largo uso di quest'attributo, partendo dal menu (differenziato in funzioni per l'utente e l'amministratore), fino alla costruzione condizionale delle funzioni disponibili sulle tabelle di dati.

Per esempio, nella fase di ricerca dell'esperimento, l'utente può cancellare o meno un esperimento selezionato sulla base del fatto di essere o meno l'autore dello stesso. Chiaramente è stata costruita una logica applicativa che sulla base anche di complesse condizioni logiche setta opportunamente le variabili booleane utilizzate negli attributi render, mostrando ,quando necessario, determinati contenuti.

Datatable

Nell'ambito della Web application realizzata, si è più volte posto il problema di visualizzare un elenco di elementi in forma tabellare, come nel caso degli esperimenti o degli utenti registrati.

L'implementazione JSF ha previsto l'utilizzo del potentissimo tag *<h:dataTable>*, il quale permette di specificare la lista degli elementi da visualizzare ed eventualmente le informazioni di formattazione. Inoltre, ciascuna colonna viene definita con il tag *<h:column>*, all'interno del quale ne viene specificato un'eventuale intestazione ed il relativo contenuto.

E' possibile dotare ogni riga di una datatable di colonne contenenti commandlink o command button, mostrati o meno a seconda del valore associato ai relativi attributi render.

Nella nostra applicazione un esempio di ciò è presente nella pagina che mostra gli esperimenti

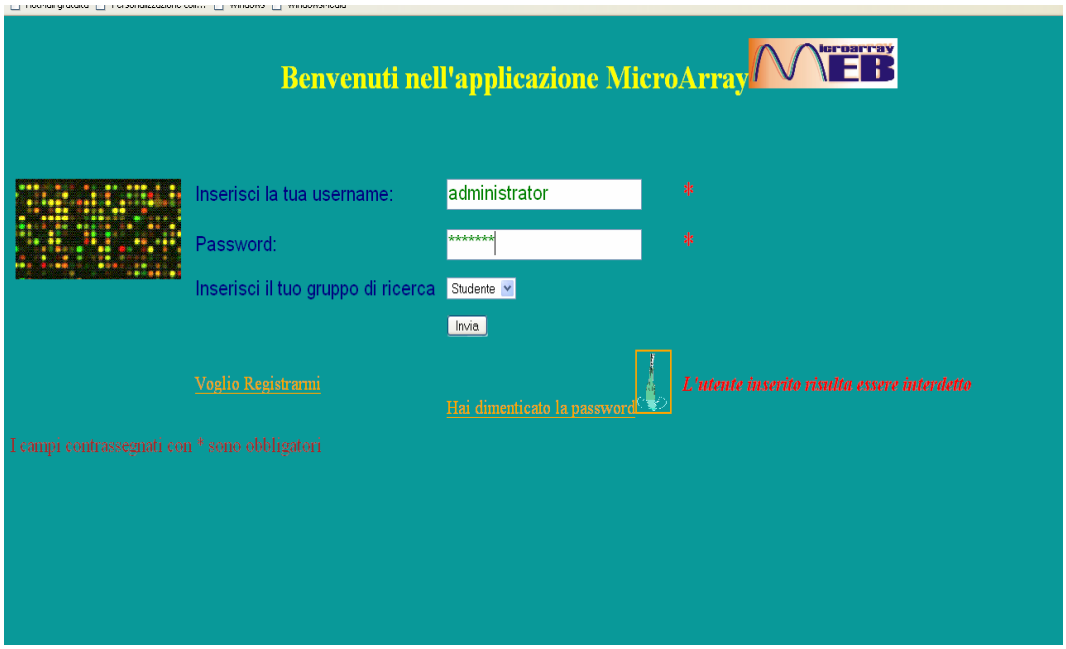
```
<h:dataTable
   value="#{experimentsTable.experiments}"
   var="experimentItem"  styleClass="tableList" headerClass="tableListHeader"
   rowClasses="tableListCustid,tableListName"
   binding="#{experimentsTable.experimentDataTable}" rows="5">

   <h:column>
      <f:facet name="header">
         <h:outputText value="#{msg.PlatformName}" />
      </f:facet>
      <h:outputText value="#{experimentItem.platformName}" />
   </h:column>
……….
   <h:column >
      <f:facet name="header">
         <h:outputText value="#{msg.DeleteExp}" />
      </f:facet>
```

```
            <h:commandButton
value="#{msg.Delete}"action="#{experimentsTable.deleteExperiment}"          onclick="return
confirm('Eliminando l'esperimento saranno eliminati anche i dati sui campioni.Continuare?');"
rendered="#{experimentsTable.delete}"/>
            </h:column>
        </h:dataTable>
```

Dall'esempio si osserva che l'attributo *value* specifica una lista definita nel bean experimetstable che contiene l'elenco degli esperimenti, mentre l'attributo *var* definisce il nome della variabile che identifica ciascun elemento all'interno della lista.

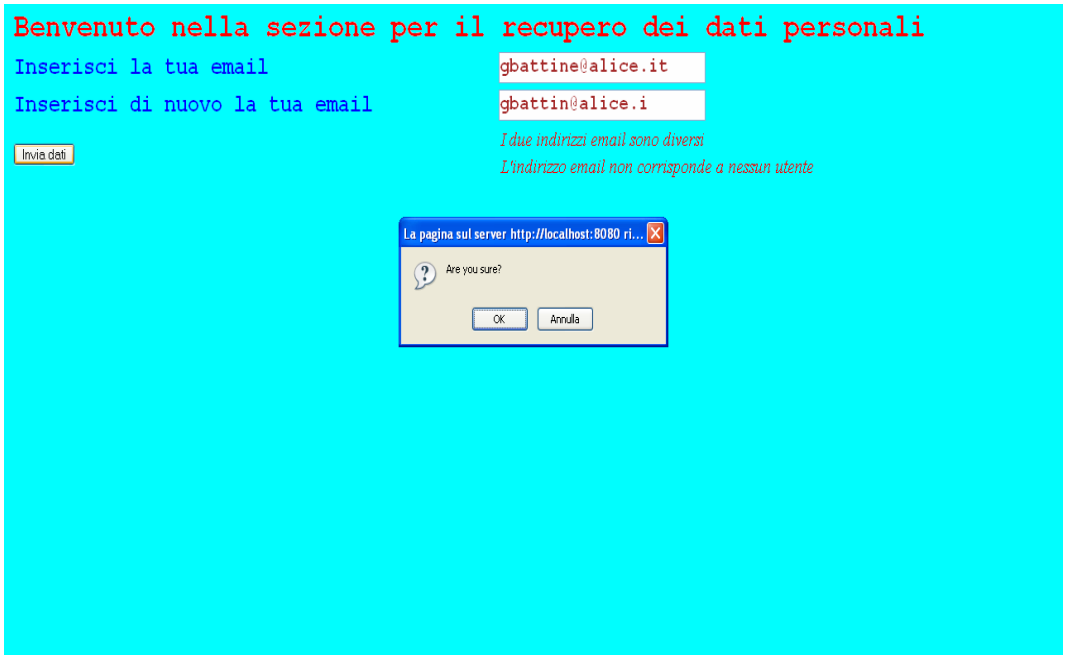
Nel momento in cui si clicca sulla descrizione dell'esperimento per poterne visualizzare le informazioni, l'oggetto corrente specificato nell'attributo *var* viene trasferito tra i parametri della request, per cui sarà disponibile alla pagina di visualizzazione.

5.4 Screenshots dell'applicazione



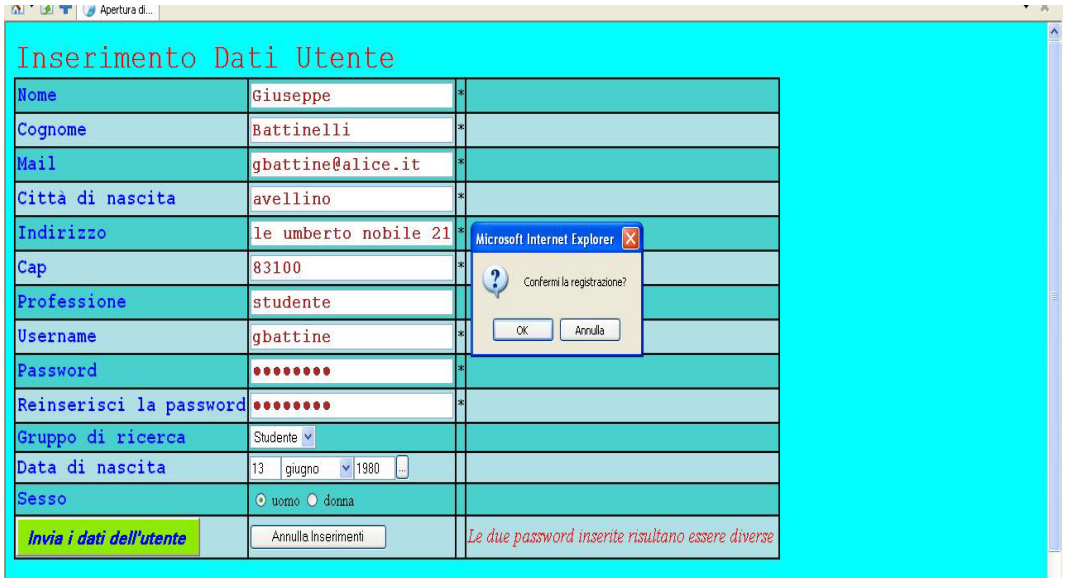
L'immagine precedente rappresenta la pagina di Login dell'applicazione, quella che permette all'utente di autenticarsi e di accedere a servizi ulteriori, quali la registrazione e la richiesta dei dati di accesso in seguito a dimenticanza o smarrimento.

In particolare abbiamo evidenziato una delle possili situazioni che possono verificarsi in seguito al controllo dei dati di accesso dell'utente, quali la possibile condizione di stato interdetto, che può essere dovuto o ad un errato inserimento della password per 3 volte consecutive o per eventuali motivi amministrativi.



Questa pagina è la pagina per il recupero dei dati personali, nella quale inserendo il proprio indirizzo email l'utente avrà la possibilità di ricevere telematicamente i dati di accesso smarriti.

Anche in questa pagina sono presenti dei controlli sul formato dell'email e sull'uguaglianza delle 2 email inserite.



La pagina precedente dà al visitatore (utente non ancora registrato) la possibilità di registrarsi, inserendo i propri dati personali.

Esistono numerosi controlli sui dati inseriti, tra cui quello sull'uguaglianza delle 2 password che mostriamo nell'immagine precedente.

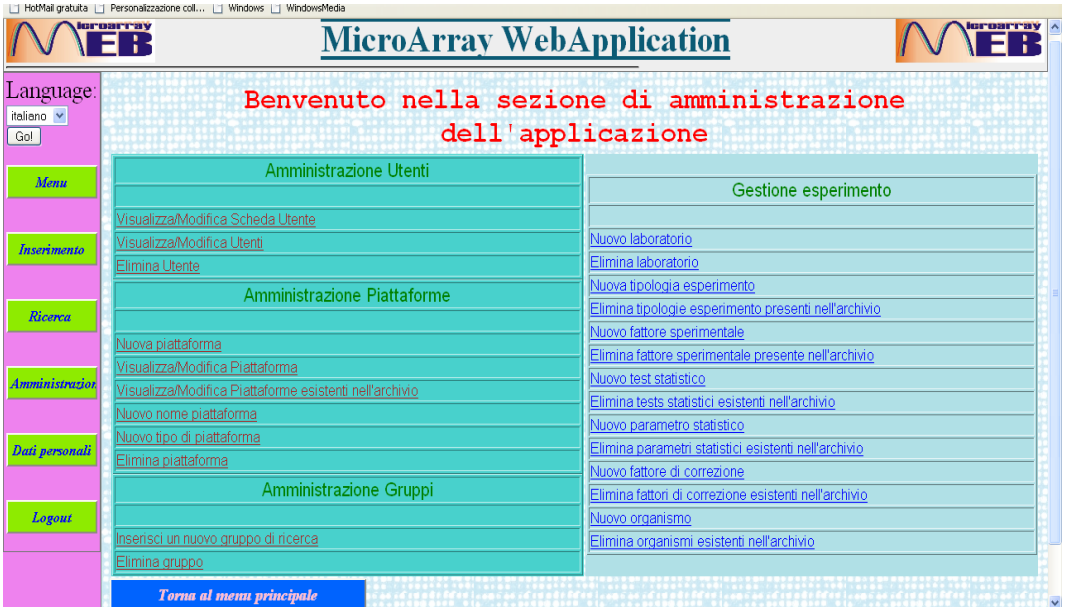
Come si può notare facilmente, questi primi screenshots hanno un formato "libero" per quanto riguarda l' impaginazione dei contenuti.

Infatti il framework Tiles, utilizzato per la creazione di una struttura unica delle pagine, è utilizzato esclusivamente per le pagine dell'applicazione, quelle visibili ad utenti registrati.

E' chiaro che queste 3 pagine mostrate in precedenza sono "esterne" all'applicazione e possono essere consultate anche da semplici visitatori.

La pagina precedente rappresenta il menu dell'applicazione, con la possibilità di accedere all'inserimento dell'esperimento, alla ricerca e alla modifica dei dati di accesso.

Inoltre è disponibile la modalità amministrazione, poiché abbiamo effettuato l'autenticazione da amministratore.

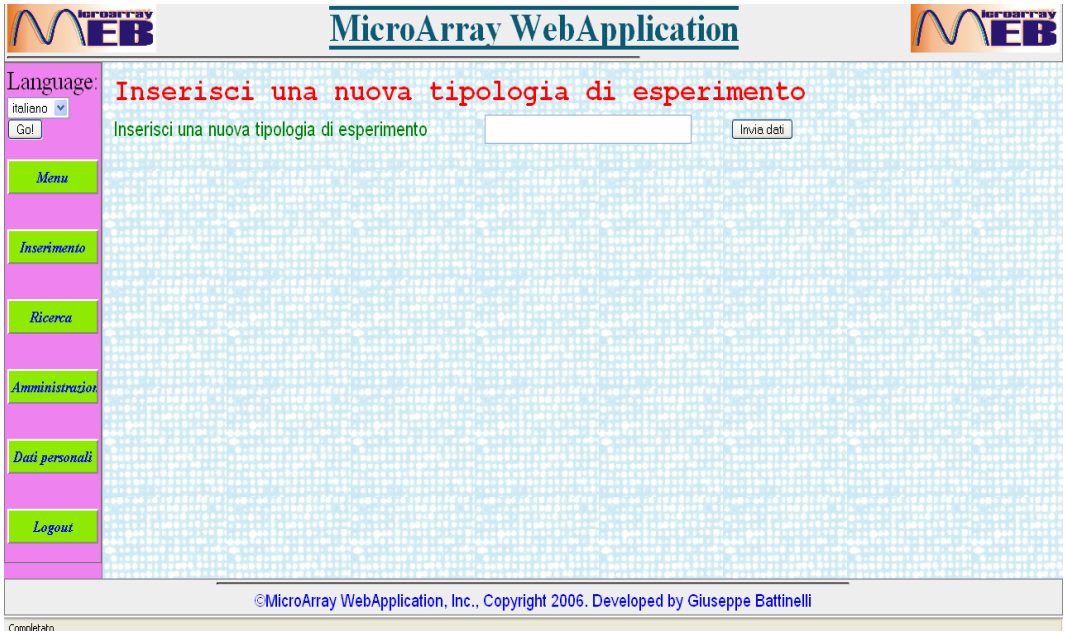


La pagina precedente rappresenta la sezione di amministrazione, il cui utilizzo è concesso al solo amministratore.

Da qui l'amministratore può accedere alla modifica di tutti i valori presenti nell'applicazione, per rendere l'applicazione facilmente manutenibile e aggiornabile.

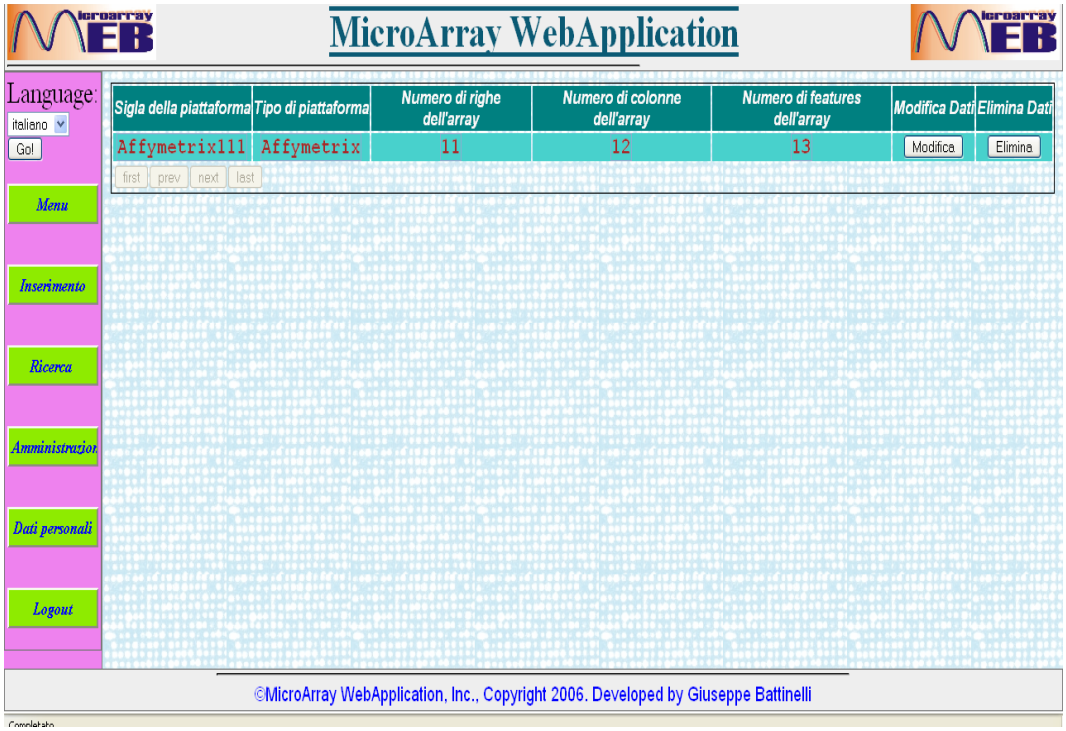
In questa pagina è presente anche una sezione di gestione degli utenti e delle piattaforme, per concedere all'amministratore un totale e completo controllo dell'applicazione.

Alcuni screenshots riguardanti questa sezione sono i seguenti:



Questa pagina rappresenta il prototipo della pagina usata nell'applicazione per inserire un nuovo fattore dell'esperimento.

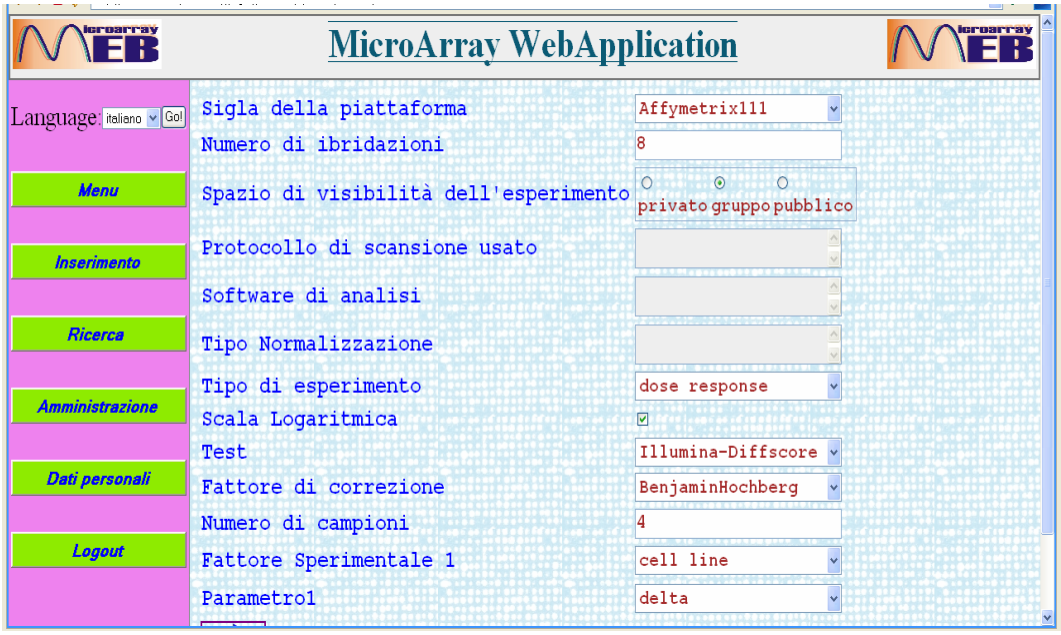
Questa pagina rappresenta il prototipo della pagina usata nell'applicazione per eliminare un fattore dell'esperimento.

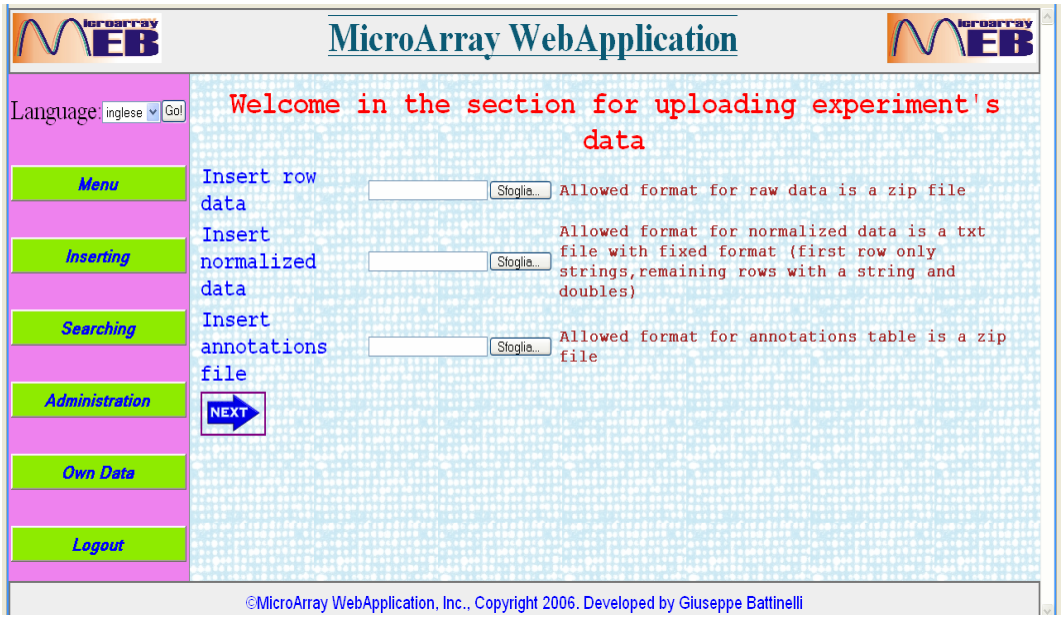


Da questa pagina l'utente può controllare i dati delle piattaforme inserite, con la possibilità di modificare i dati relativi ad una piattaforma o eliminarli del tutto dall'applicazione.

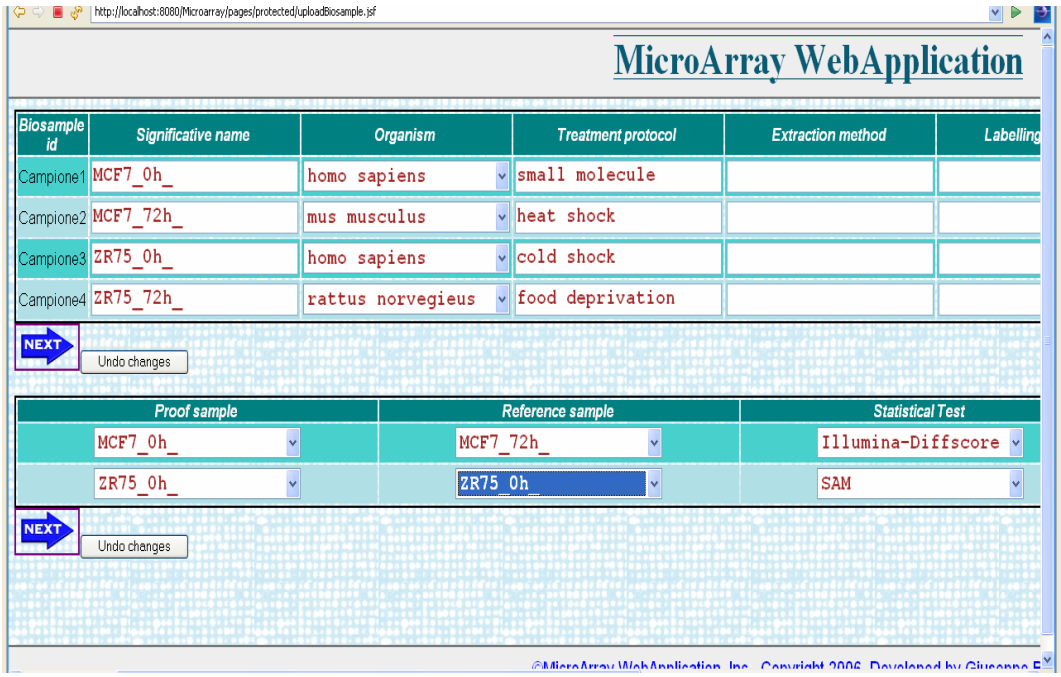
Quelli che mostriamo adesso, sono alcuni screenshots riguardanti la fase di inserimento dell'esperimento, alcuni mostrati in lingua italiana, altri nella modalità inglese, per mostrare l'internazionalizzazione dell'applicazione.

MicroArray WebApplication
Language: italiano Go!
Menu
Inserimento
Ricerca
Amministrazione
Dati personali
Logout
Benvenuto nella pagina di inserimento dei dati dell'esperimento
Breve descrizione dell'esperimento
* Devi riempire questo campo.
Autore
Giuseppe
*
Laboratorio dell'esperimento
cnr
Contatti
Links e url
Controlli di qualità
Numero Fattori Sperimentali
1
Numero di analisi svolte
2
*
Numero di parametri statistici
1
Annulla Inserimenti
©MicroArray WebApplication, Inc., Copyright 2006. Developed by Giuseppe Battinelli


MicroArray WebApplication
Language: italiano Go!
Menu
Inserimento
Ricerca
Amministrazione
Dati personali
Logout
Sigla della piattaforma
Affymetrix111
Numero di ibridazioni
8
Spazio di visibilità dell'esperimento
privato gruppo pubblico
Protocollo di scansione usato
Software di analisi
Tipo Normalizzazione
Tipo di esperimento
dose response
Scala Logaritmica
Test
Illumina-Diffscore
Fattore di correzione
BenjaminHochberg
Numero di campioni
4
Fattore Sperimentale 1
cell line
Parametro1
delta

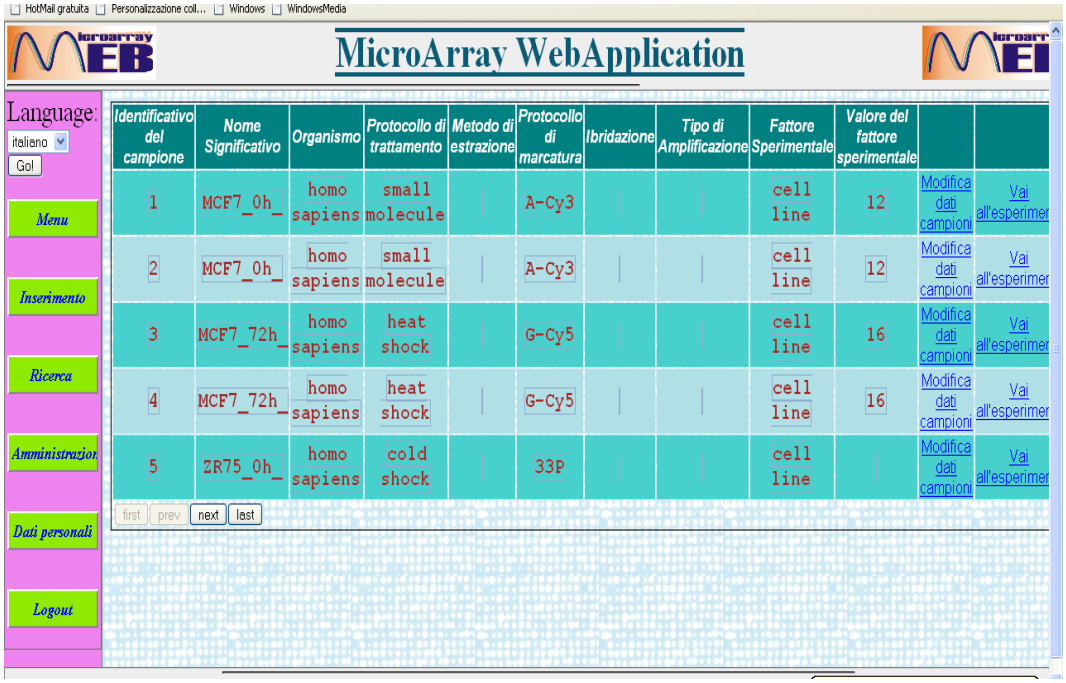


Lo screenshot precedente mostra la fase di inserimento dei dati dell'esperimento, spiegando con dei brevi messaggi di testo il tipo di formato accettato per ogni campo.

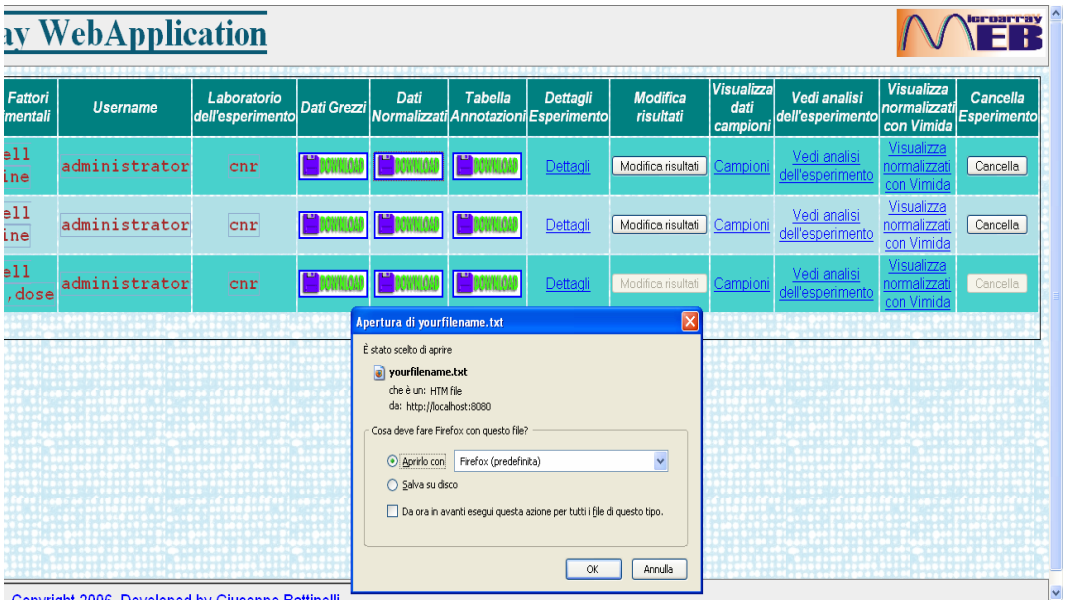


Lo screnshot precedente mostra, invece, la delicata fase di inserimento dei campioni e delle analisi statistiche effettuate sui dati.

E' in realtà una delle fasi più importanti, in quanto nella seconda tabella è possibile descrivere "gli incroci" tra i campioni inseriti nella prima tabella che rappresentano ognuno un'analisi sui dati, effettuata con un particolare Test statistico e che porterà (non è evidenziato nella pagina perché necessita di scorrimento laterale) all'inserimento delle particolari liste dei geni regolati riguardanti la particolare analisi.

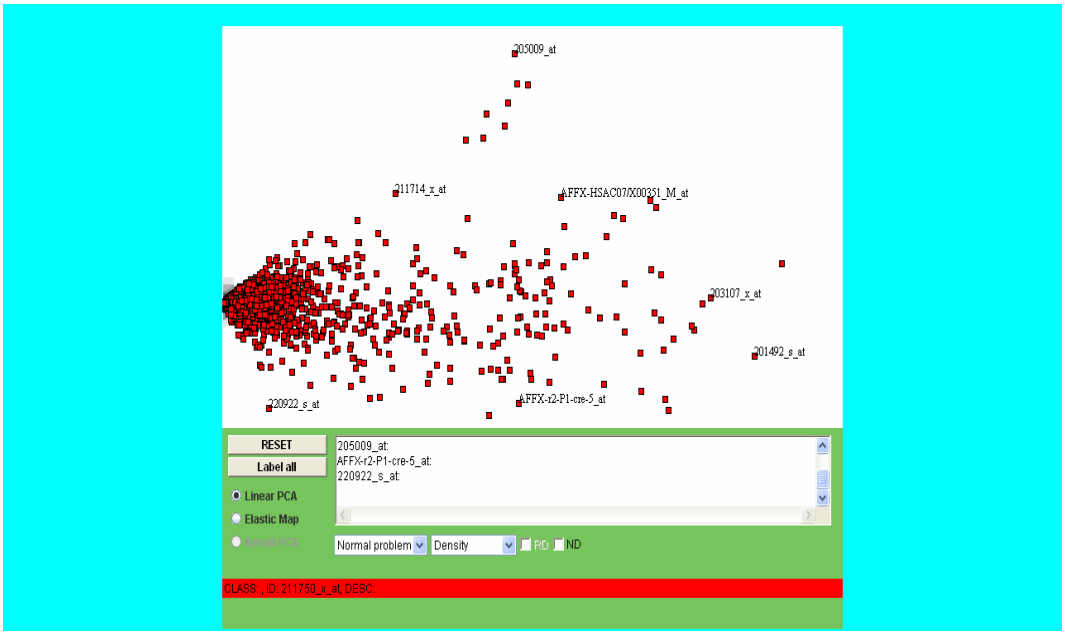


In questa pagina è possibile analizzare i campioni dell'esperimento ed effettuare, se necessario, delle modifiche.



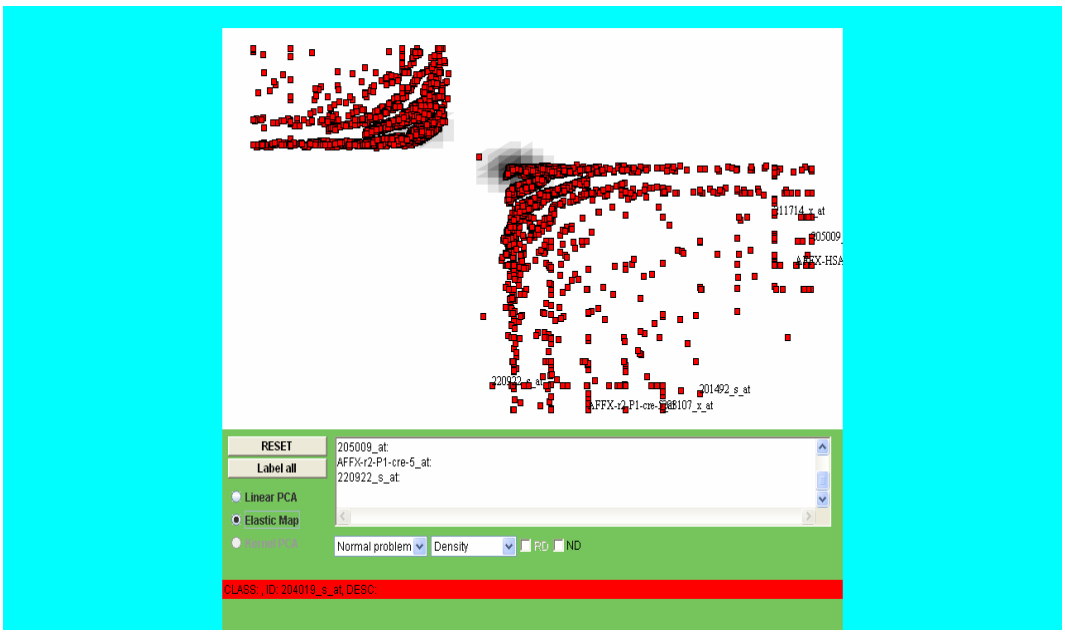
Questa pagina è fondamentale per l'applicazione, in quanto consente di analizzare tutti gli esperimenti presenti nell'applicazione, con la possibilità di lanciare l'applet Vimida per visualizzare graficamente i risultati, modificare un esperimento, osservarne i dettagli e scaricarne i files.

Questa è la pagine dei dettagli dell'esperimento, la quale ci mostra ogni campo di quest'ultimo, con la possibilità di effettuare modifiche dove necessario.

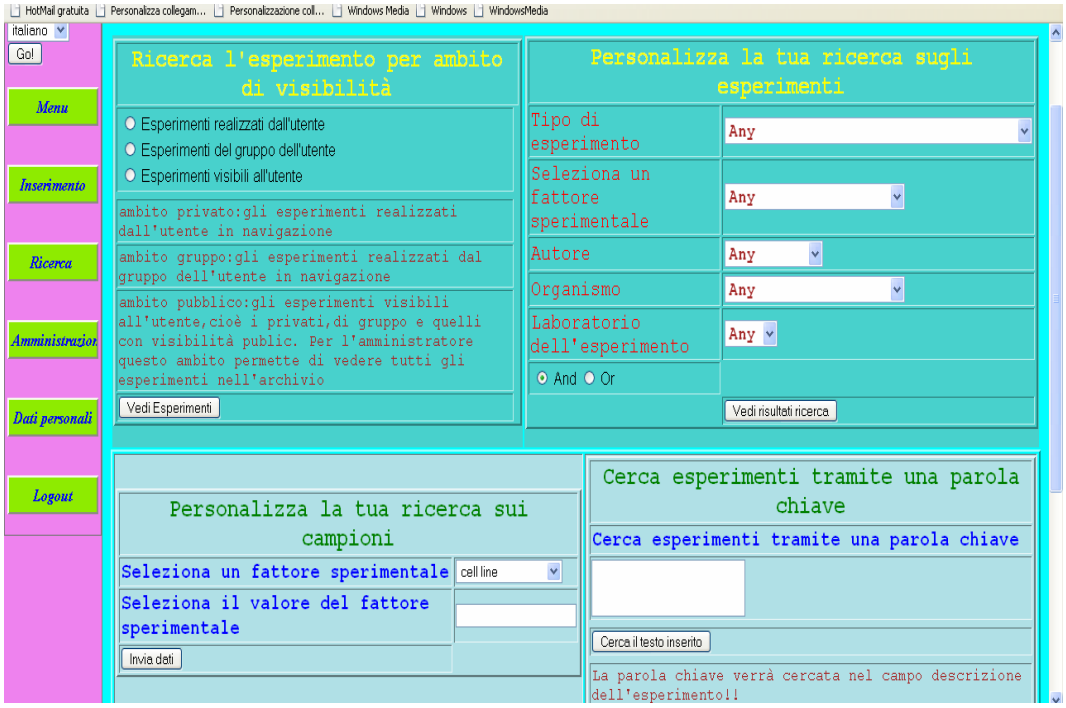


L'applet Vimida integrata nell'applicazione consente la visualizzazione grafica dei dati normalizzati dell'applicazione, con la possibilità di utilizzare 2 diversi algoritmi di classificazione al riguardo.

L'immagine precedente mostra l'analisi ottenuta con la PCA o Linear Map



mentre quest'altra mostra un'analisi effettuata con l'Elastic Map.



Questa pagina, invece, rappresenta il nucleo delle funzionalità dell'applicazione in fase di ricerca. Da qui possono partire tutte le richieste dell'utente/amministratore al fine di ricavare in maniera utile ed immediata i dati di interesse.

5.5 Internazionalizzazione (I18N)

Attraverso il meccanismo dell'Internazionalizzazione (I18N), è possibile visualizzare il testo presente nella Web application in lingue diverse, in relazione alla localizzazione geografica da cui si collega l'utente oppure sulla base delle impostazioni della lingua del browser. Nel caso dell'applicazione realizzata, si è previsto il supporto per la lingua italiana, considerata di default, e per la lingua inglese.

In questo modo, senza la necessità di dover sviluppare due volte la medesima Web application, è possibile comunque garantire l'utilizzo di quest'ultima a persone di lingua differente.

Per quanto riguarda l'implementazione sono stati realizzati due coppie di due Resource Bundle ciascuna.

In particolare è stato realizzato un resource bundle chiamato *messageResource*, che contiene i messaggi delle pagine web nelle 2 versioni, italiano e inglese e un altro resource bundle, chiamato *application* che allo stesso modo contiene i messaggi (nelle 2 lingue) generati dalla logica dell'applicazione, cioè dai backing beans.

Ogni resource bundle è costituito da una serie di coppie "key-message", attraverso le quali è possibile visualizzare un certo messaggio semplicemente specificando la chiave ad esso associata.

Un esempio è il seguente

messageResource.properties

firstNameInput=Nome dell'utente

lastNameInput=Cognome dell'utente

...

messageResource_en.properties

firstNameInput=User's name

lastNameInput=User's lastname

...

Come si può osservare, in entrambi i file (con estensione *properties*) le chiavi sono perfettamente identiche mentre i messaggi sono in lingua diversa.

Inoltre, i file stessi hanno praticamente il medesimo nome a meno del file in lingua inglese che ha il suffisso "*en*".

Tipicamente, il file che non ha alcun suffisso è quello relativo alla lingua di default mentre i file associati alle altre lingue devono avere lo stesso nome del primo ma seguito da un suffisso che identifica la lingua.

Un esempio d'uso del precedente resource bundle è

```
<h:outputText value="#{msg.FirstNameInput}" styleClass="outputText"/>
```

Per la registrazione dei Resource Bundle è stato necessario specificare la localizzazione (*Locale*) di default e quella supportata, oltre al nome del file contenente i messaggi, nel faces-config.xml dell'applicazione.

```
<application>
  <message-bundle>microarray.bundle.messageResource</message-bundle>
  <message-bundle>microarray.bundle.application</message-bundle>
  <locale-config>
   <default-locale>it</default-locale>
   <supported-locale>it</supported-locale>
   <supported-locale>en</supported-locale>
  </locale-config>
 </application>
```

Per quanto riguarda l'utilizzo dell'internazionalizzazione all'interno delle pagine dell'applicazione è previsto l'uso del tag *<f:loadBundle>* in cui l'attributo *basename* specifica il nome del Resource Bundle mentre *var* permette di assegnargli un nome riferimento all'interno della pagina JSP.

Un esempio è

```
<f:loadBundle basename="microarray.bundle.messageResource" var="msg"/>
```

Per quanto riguarda i messaggi prodotti dai backing bean abbiamo deciso di realizzare un metodo che, richiamato quando necessario, consentisse di mostrare un messaggio prodotto dai beans.

```
protected void appMessage(String clientId,String key){
        String text=null;
        try{
                ResourceBundle
            bundle=ResourceBundle.getBundle("microarray.bundle.application",
            getFacesContext().getViewRoot().getLocale());
            text=bundle.getString(key);
        }
        catch(Exception e){
            text="???"+key+"???";
        }
        getFacesContext().addMessage(clientId,new FacesMessage(text));
}
```

Grazie a questo metodo è possibile, in un bean, mostrare un messaggio del bundle, in seguito ad un particolare evento, semplicemente con

```
appMessage(null,"UtenteAttesa");
```

grazie al quale verrà mostrato il messaggio corrispondente alla chiave UtenteAttesa del resource bundle application,nel locale della sessione.

5.6 Validazione

Per la validazione dei dati immessi nei form da parte dell'utente JSF mette a disposizione un proprio meccanismo per la validazione e per la sua estensione.

5.6.1 Custom validator

JavaServer Faces mette a disposizione soltanto tre validatori standard che permettono di valutare se un certo valore ha una certa lunghezza entro un minimo ed un massimo prefissati oppure se il medesimo valore rientra in un certo range.

Per quanto riguarda l'obbligatorietà di un campo, quest'ultima viene specificata mediante l'attributo *required* del tag associato al componente in questione.

Un esempio di obbligatorietà di un campo può essere la richiesta della password in fase di registrazione, come mostrato di seguito

```
<h:outputText value="#{msg.PasswordInput}" styleClass="outputText"/>
<h:inputSecret value="#{user.password}" id="passwordInput" required="true"
        styleClass="inputText">
<f:validateLength maximum="20" minimum="5"/>
</h:inputText>
```

Esempio di campo obbligatorio

Per quanto riguarda il controllo che un valore di un campo debba avere una lunghezza minima prefissata, si può fare riferimento al tag *<f:validateLength>* utilizzato in fase di registrazione per la richiesta della password, per la quale è stata fissata una lunghezza minima di 5 caratteri alfanumerici ed una massima di 20.

Considerando il numero limitato di controlli standard che possono essere seguiti, il framework mette a disposizione due meccanismi per la realizzazione di ulteriori validatori. Il primo consiste nel realizzare un metodo che esegua la validazione all'interno di un backing bean, il secondo prevede lo sviluppo di una classe che implementi l'interfaccia *Validator*, alla quale può poi essere associato o meno un tag. Tipicamente la prima soluzione è strettamente limitata all'uso dello specifico backing bean e non permette di usare il validatore su più componenti che prevedono lo stesso tipo di controllo, a meno di non replicare il metodo di validazione nei vari backing beans. Per questo motivo si è preferita la seconda soluzione grazie alla quale un unico validatore può essere facilmente associato a più componenti.

L'implementazione in JSF ha previsto lo sviluppo del seguente validator :

EmailValidator : permette di valutare se il formato di un indirizzo email sia o meno corretto;

5.6.1.1. EmailValidator

Questo validatore viene utilizzato nella fase di registrazione di un utente oppure nel momento in cui quest'ultimo richiede l'invio di una mail con i propri dati di accesso, specificando il proprio indirizzo email.

La classe EmailValidator implementa l'interfaccia Validator della quale esegue l'overriding del metodo *validate()*, per valutare se l'indirizzo email immesso dall'utente abbia o meno un formato valido.

Per fare ciò ci siamo serviti di un potentissimo strumento messo a disposizione da Java 1.5, le espressioni regolari.

```
public class EmailValidator implements Validator {

public void EmailValidator(){

}
        public void validate(FacesContext arg0, UIComponent arg1, Object object) throws
ValidatorException {
          String enteredEmail = (String)object;
          Pattern p = Pattern.compile(".+@.+\\.[a-z]+");
        Matcher m = p.matcher(enteredEmail);
        boolean matchFound = m.matches();
        if (!matchFound) {
                  FacesMessage message = new FacesMessage();
                  message.setDetail("Email non valida");
          message.setSummary("Il formato dell'email inserita non è
                  valido");
          message.setSeverity(FacesMessage.SEVERITY_ERROR);

        throw new ValidatorException(message);
      }

}
```

Per poter effettuare il controllo, all'interno del metodo validate() viene utilizzato lo strumento delle Regular Espression (Espressione regolare) fornito dal linguaggio Java. In particolare si specifica quale sia il pattern a cui deve essere conforme un indirizzo email per essere considerato corretto e si valuta se il valore immesso dall'utente corrisponda ad esso.

Si osserva che se il valore immesso dall'utente non è conforme al pattern specificato, viene sollevata un' eccezione la quale contiene il messaggio da visualizzare all'utente. Tale messaggio viene prelevato dal Resource Bundle. Il validatore ha ovviamente previsto una registrazione all'interno del file di configurazione, così come tutti gli elementi personalizzati che si creano con JSF.

```
<validator>
        <validator-id>EmailValidator</validator-id>
        <validator-class>giu.EmailValidator</validator-class>
</validator>
```

Il suo inserimento in una pagina Jsp avviene tramite il tag <f:validator> della libreria core in questo modo

```
<h:outputText value="#{msg.MailInput}" styleClass="outputText"/>
<h:inputText value="#{user.email}"  id="email" required="true"
        styleClass="inputText">
    <f:validator validatorId="EmailValidator" />
 </h:inputText>
```

5.7 Connessioni al database e "teoria del pooling"

Nelle applicazioni Java Enterprise è considerata una procedura non pulita instanziare connessioni verso il database da soli. La via giusta richiede un pool di connessioni verso il database, nella forma di un JDBC DataSource, che viene fornito dal container (Tomcat, Jetty, WebLogic). Il container è responsabile di mantenere un certo numero di connessioni verso il database e tutto il nostro codice per ottenere una connessione deve solo fare la lookup del DataSource tramite JNDI.

Molte applicazioni web necessitano di accedere al database attraverso un driver JDBC per supportare le funzionalità offerte dall'applicazione. Tomcat 5.5 offre esattamenente lo stesso supporto offerto dagli application servers della J2EE, così che applicazioni basate su connessioni al database sviluppate su Tomcat, usando il servizio del datasource gireranno senza alcuna modifica su qualsiasi server J2EE.

Teoria del pooling.

La maggior parte dei DataSource si basa su pool di connessioni al database, quindi è difficile parlare di uno senza aver prima parlato dell'altro. Il "Pooling" è una maniera di gestire le risorse che sono costose da creare e in costante richiesta, ma usate soltanto per un breve periodo di tempo. La maggior parte degli sviluppatori Java è familiare con i pool di connessioni al database, o almeno con pool di thread.

In linguaggi che non forniscono la Garbage Collection, come C e C++, non è raro usare pool di memoria raw.

La premessa da fare sul "pooling" è abbastanza semplice: *invece di instanziare direttamente una risorsa* -- chiamando new -- *il codice prende in prestito una risorsa da un repository centralizzato*.

Il codice client usa la risorsa e la restituisce al repository. Il pooling di oggetti, metaforicamente parlando, può essere considerato come il noleggio di una autovettura: il codice client non conosce l'origine della risorsa, nè cosa succede quando la restituisce, nè se otterrà sempre lo stesso oggetto ogni volta che lo richiede. Il repository è responsabile per il ciclo di vita degli oggetti risorsa, cioè istanzia un certo numero di oggetti risorsa e li rende disponibili per l'uso da parte dell'utente e successivamente li rinizializza. Il pool è esaurito quando tutte le risorse sono in prestito. Cosa succede quando il codice richiede una risorsa da un pool prosciugato è un dettaglio relativo all'implementazione del pool: l'operazione di richiesta dal pool può essere bloccante (ovvero non ritorna) finche una risorsa è disponibile, ritornare null o lanciare un eccezione.

Quello che va tenuto in conto quando si realizza un'applicazione web più che una desktop sono le complicazioni che quest'ultima può dare nel normale funzionamento di tutti i giorni.

Nelle più classiche applicazioni client-server, scritte in modo accurato, la connessione con il database non dovrebbe quasi mai generare problemi.

Normalmente ,una volta autenticato l'utente, viene aperta una connessione che rimane viva sino a quando l'applicazione viene chiusa. Sono veramente pochissime le possibilità che queste connessioni rimangano aperte, provocando complicazioni sul database.

Ma nel caso di applicazioni che viaggiano attraverso il web, gestire la chiusura di una connessione è una operazione molto più complessa. A differenza delle applicazioni classiche, infatti, la chiusura della connessione viene lasciata all'utente che, al termine della sua sessione di lavoro, dovrebbe esplicitamente effettuare una operazione di logout dal database.

Ma non è solo questo l'unico problema legato a questo argomento.

Il rischio di timeouts, dovuti ad attese prolungate, che possano anche derivare da momenti di lentezza della rete o da una più bassa velocità di trasferimento, possono rivelarsi pericolose, perchè non permetterebbero la corretta gestione delle risorse che sono state aperte.

Ed ancora, la possibilità che un elevatissimo numero di utenti possano accedere alla stessa applicazione web in contemporanea può comunque rivelarsi problematico, soprattutto se il database ha un numero limitato di connessioni contemporanee possibili.

Il primo istinto sarebbe quello di risolvere questi problemi semplicemente ricreando la connessione al database prima di ogni operazione e chiuderla una volta terminata la singola operazione.

Un po' di codice ben scritto permetterebbe anche di gestire tutti i possibili errori derivanti dalla rete (timeouts, problemi di comunicazione) e di fornire quindi una buona garanzia di non lasciare nessuna connessione o risorsa aperta.

Sebbene questo possa sembrare un ottimo escamotage, si genererebbe un differente tipo di problema, legato principalmente al fatto che *la connessione è una operazione piuttosto lenta* e quindi ricrearne una nuova ogni volta può portare ad un forte degrado delle prestazioni dell'applicazione stessa.

Soluzione adottata nella web application presentata.

La tecnica da noi implementata sarò quella della realizzazione di un Datasource,o fornitore di connessioni.

Come anticipato prima, per questo scopo ci siamo "serviti" di un servizio offerto da Tomcat al riguardo.

Il datasource di default offerto da Tomcat è basato sul connection pool delle DBCP del progetto Jakarta Commons. In realtà avremmo potuto usare qualsiasi altro connection pool che implementasse javax.sql.DataSource.

La premessa al procedimento usato passa attraverso una riflessione sul funzionamento di Tomcat.

Tomcat 5 fornisce un' istanza di implementazione JNDI di un contesto iniziale per ogni web application che gira sotto esso.

L' InitialContext è configurato non appena la web application è caricata ed è reso disponibile a tutti i componenti della web application (solo per un accesso in lettura).

Tuti i parametri e le risorse configurate sono poste nella porzione java:comp/env del namespace JNDI, per cui un tipico accesso ad una risorsa (nel nostro caso ad un datasource JDBC) passa attraverso

```
try {
        Context ctx = new InitialContext();
        if (ctx == null) {
                throw new Exception("No Context");
        }
        ds = (javax.sql.DataSource) ctx.lookup("java:comp/env/MysqlJNDI");
}
catch (Exception e) {
        e.printStackTrace();
```

}

In cui otteniamo il nostro contesto e cerchiamo il datasource dell'applicazione.

Come già detto in precedenza una connection pool crea e gestisce un pool di connessioni a un database.

L'idea fondamentale è che riciclare e riusare connessioni ad un database già esistenti sia molto più efficiente che aprire ogni volta una nuova connessione.

Problema del pool di connessioni.

In realtà c'è un problema che si potrebbe avere con il pool di connessioni. Una web application dovrebbe chiudere esplicitamente i ResultSets, gli Statements e le Connections aperte, perché il non farlo potrebbe renderli inutilizzabili per un futuro riuso, causando il cosiddetto "connection pool leak".

Ciò può portare al fallimento dell'apertura di una nuova connessione se non dovessero essercene più di disponibili.

In realtà esiste una soluzione che noi abbiamo adottato nella nostra web application.

Le jakarta Commons-DBCP possono essere configurate per tenere traccia e recuperare queste connessioni al db abbandonate, rendendo possibile non solo il loro recupero ma anche la generazione di uno stack trace del codice che ha aperto queste risorse senza chiuderle mai.

Configurazione del servizio

La parte fondamentale della configurazione di questo servizio consiste nell'inserirlo nel contesto della web application, nel server.xml di Tomcat.

In particolare nel context dell'applicazione Microarray è stato necessario aggiungere

```
<Resource name="MysqlJNDI"
          auth="Container"
          type="javax.sql.DataSource"
          maxActive="100"
          maxIdle="30"
          maxWait="100000"
          username="root"
          password="shevagol"
          driverClassName="com.mysql.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/Microarray?autoReconnect=true"
          removeAbandoned="true"
          removeAbandonedTimeout="30"
         lobAbandoned="true">
</Resource>
```

Analizziamo i principali attributi:

maxActive: Massimo numero di connessioni al db nel pool, compatibilmente con l'attributo max_connections di Mysql. Il valore 0 indica nessun limite.

maxIdle: Massimo numero di connessioni inattive da mantenere nel pool. Il valore -1 indica nessun limite.

maxWait: Massimo tempo di attesa per la ricezione di una connessione al db(in secondi) prima del lancio di un'eccezione. Il valore -1 attende senza limiti.

driverClassName: Nome del driver utilizzato per la connessione.

url: L'url per la connessione JDBC al dB.

Il parametro autoReconnect=true fa sì che il *com.mysql.jdbc.Driver* driver si riconnetterà automaticamente se mysql chiude la connessione, cosa che di default avviene dopo 8 ore di connessione inattiva.

Importanza fondamentale hanno gli ultimi parametri, per quanto detto in precedenza.

- *removeAbandoned="true"*

fa sì che le connessioni al db abbandonate siano rimosse e riciclate.Questo vuol dire che quando le connessioni disponibili si stanno esaurendo DBCP recupererà e riciclerà tutte le connessioni al db che trova. Il default è false.

- *removeAbandonedTimeout="30"*

Serve per settare il numero di secondi che una connessione al db deve rimanere inattiva prima di essere considerata abbandonata.

- *logAbandoned="true"*

Questo attributo può essere settato a true se si vuole che le DBCP generino uno stack trace del codice che ha abbandonato le connessioni senza chiuderle.

L'ultimo passaggio nella configurazione del servizio è la registrazione della resource nel web.xml dell'applicazione web.

```
<!-- configurazione datasource -->
        <resource-ref>
                <description>DB Connection</description>
                <res-ref-name>jdbc/microarray</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
        </resource-ref>
```

5.8 Inserimento sicuro dei dati e uso delle "transazioni".

Come più volte evidenziato e sottolineato, la nostra applicazione web dovrà garantire "in primis" dei requisiti di sicurezza.

Nel primo paragrafo abbiamo mostrato le tecniche usate per garantire l'accesso alle pagine solo se autorizzato e i meccanismi di gestione della chiusura anticipata del browser.

Un altro requisito fondamentale riguarda la sicurezza con la quale va effettuato l'inserimento dei dati.

L'inserimento di un esperimento nell'archivio passa attraverso numerose pagine web ed è importante garantire che, in caso di uscita anticipata dall'applicazione o di problemi accidentali quali guasti di rete, elettrici od altro , l'inserimento nell'archivio avvenga solo se completato precedentemente dall'utente.

E' importante evitare che l'esperimento possa essere inserito solo "parzialmente", rendendo così la nostra base dati "sporca".

La soluzione a questo problema passa per il concetto di transazione.

Cosa si intende per transazione?

La transazione è un' unità elementare di lavoro sulla base di dati di cui si vuole garantire le cosiddette proprietà ACIDE (o Acid dall'inglese).

I DBMS prevedono meccanismi per gestire la definizione e l'esecuzione di transazioni.

Sintatticamente un transazione è delimiatata dai comandi **begin transaction** e **end transaction**; all'interno possono comparire i comandi di **commit work** e **rollback work**.

Un comando di **commit** trasferisce gli effetti della transazione sulla base di dati

Un comando di **rollback** (*abort*) annulla gli effetti della transazione e lascia inalterata la base di dati.

Le proprietà ACIDE (dall'acronimo ACID) che una transazione ha il dovere di rispettare possono essere così riassunte:

1. Atomicità: ciascuna transazione è un'unità indivisibile di esecuzione
2. Consistenza: l'esecuzione di una transazione non deve violare l'integrità della base di dati.
3. Isolamento: il risultato dell'esecuzione di una transazione deve essere indipendente dall'esecuzione delle altre transazioni.
4. Persistenza o Durability: gli effetti dell'esecuzione di una transazione andata in commit non devono essere persi.

Per venire incontro a questi requisiti abbiamo deciso di creare un contenitore di oggetti (in questo caso i prepared statements che costituiscono le query create dall'utente durante l'inserimento dell'esperimento) che viene riempito pagina dopo pagina con gli statements generati dagli inserimenti dell'utente. Questo contenitore viene riempito ma le query non vengono eseguite.

Giunti all'ultima pagina degli inserimenti entra in gioco la classe *"QueryManager"*, creata con lo scopo di leggere questo contenitore (arrayList) di statements ed eseguire ogni query presente. La situazione di "almeno una query non eseguita" genera il rollback della transazione, la situazione di "tutte le query eseguite correttamente" genera il commit della transazione.

Questa è la parte di codice che sintetizza quanto detto

```
conn.setAutoCommit(false);
for(int j=0;j<results.size();j++){
if((results.get(j)).equals("false"))
        conn.rollback();
}
conn.commit();
 }
```

Dove il "false" è generato da una query non correttamente eseguita.

5.9 Il framework Tiles.

Tiles è un framework mediante il quale è possibile realizzare il Presentation Layer di una Web application, separando il layout dai contenuti (contents). Mentre il layout definisce la struttura di una pagina, con la caratteristica che più pagine possono avere il medesimo layout, il contents definisce il contenuto di ciascuna di esse e può essere diverso da pagina a pagina, oppure eventualmente uguale in alcuni casi.

E' notevolmente importante poter separare l'uno dall'altro, in modo tale che apportando delle modifiche al primo non sussiste alcuna influenza sul secondo e viceversa. Il framework permette di costruire le pagine assemblando i cosiddetti "tiles", il cui significato è banalmente "mattonelle", evidenziando il fatto che sono dei componenti riutilizzabili che possono essere disposti in una struttura predefinita.

Nella maggior parte dei casi, ogni tile non è nient'altro che una pagina JSP che a sua volta può essere composta da altri tiles. E' possibile distinguerne due tipologie:

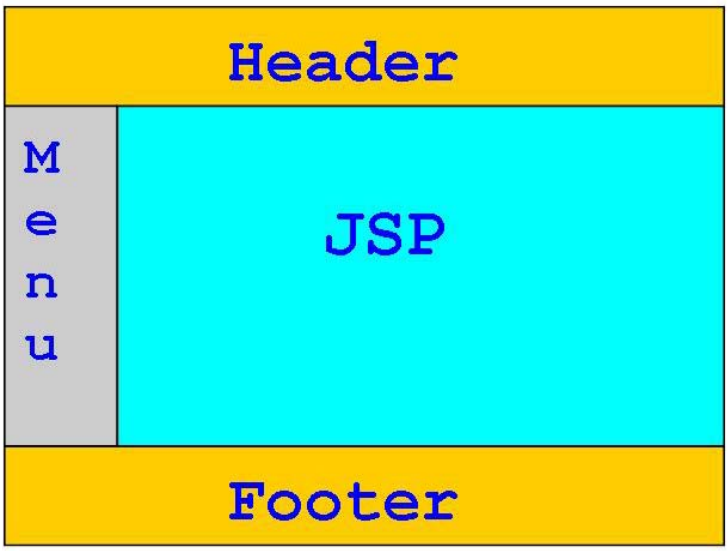
- tile Layout: tile che definisce la struttura di una pagine. Esso riceve da una pagina (tile non-Layout) il contenuto e lo dispone secondo tale struttura;
- tile non-Layout: tile che utilizza un layout (tile Layout) passando a quest'ultimo degli attributi che rappresentano il contenuto da disporre secondo una certa struttura;

Generalmente, nell'ambito di una Web application, il tile Layout è unico e definisce la struttura di tutte le pagine, mentre ciascuna pagina è definita attraverso un proprio tile non-Layout.

L'uso dei tiles può essere paragonato all'uso dei metodi in Java.

Un metodo in java è costituito da un body che ne definisce l'elaborazione e dai parametri di ingresso che specificano i dati su cui quest'ultima va effettuata. Un tile può essere considerato come un metodo, in quanto bisogna definirne la struttura e passargli dei parametri, detti attributi, che ne descrivano il contenuto.

Ciascuno di essi è memorizzato nel context del tile stesso, attraverso la classe ComponentContext facente parte del framework. Infine, ogni attributo può essere una stringa oppure tipicamente una pagina JSP. Nella maggior parte dei casi, il layout utilizzato per ciascuna pagina è proposto in figura.



Esso prevede fondamentalmente quattro parti: header, fortore, menu e body. Tipicamente, l'header ed il footer sono comuni a tutte le pagine, per cui il loro contenuto viene definito un'unica volta.

Il vantaggio che ne scaturisce è dato dal fatto che se deve essere apportata una modifica ad uno di essi, quest'ultima va eseguita un'unica volta e non per tutte le pagine della Web Application.

Il menu, invece, può essere lo stesso nell'ambito di tutta l'applicazione e quindi avrà il medesimo vantaggio di header e footer oppure può variare in relazione alle pagine visitate. Infine, il body è tipicamente diverso da pagina a pagina.

Per quanto riguarda la relazione con Jsf, Tiles non è perfettamente integrato come nel framework Struts ma va inglobato tramite l'apposita libreria.

Definitions.

I tiles possono essere descritti attraverso le cosiddette *definitions*, dichiarate in un file XML e che permettono:

- Una dichiarazione centralizzata di una pagina;
- Se si dispone di più tiles non-Layout con un contenuto uguale (es. le pagine hanno il medesimo header e footer), al posto di definire tale contenuto pagina per pagina, lo si definisce un'unica volta rendendolo comune a tutte le pagine;
- Supporto alla derivazione, descrivendo una nuova definition che ne estenda una preesistente, ereditandone le caratteristiche ed aggiungendone delle altre;

Generalmente, il file che contiene le definitions è noto come *tiles-defs.xml* e ciascuna di esse è dichiarata attraverso il tag <*definition*>, che mediante i suoi attributi permette di specificare le seguenti informazioni:

- Nome della definition per referenziarla all'interno di un tile che ne farà uso *(name)*;
- Percorso del tile Layout che descrive la struttura associata alla definition *(path)*;
- Definition da cui eseguire la derivazione *(extends);*

Per specificare gli attributi della definition, è possibile utilizzare uno o più tag <*put*>, ciascuna dei quali prevede le seguenti informazioni:

- Nome dell'attributo;
- Valore dell'attributo che può essere una stringa, una pagina JSP oppure un'altra definition se si prevede di utilizzare la composizione di più definitions;

Un esempio di *tiles-defs.xml* è quello della nostra applicazione

```
<tiles-definitions>
        <definition  name="page.template" path="/templates/topLayout.jsp"/>
        <definition  name="page.header" path="/templates/header.jsp"/>
        <definition  name="page.leftMenu" path="/templates/menu.jsp"/>
        <definition  name="page.footer" path="/templates/footer.jsp"/>
</tiles-definitions>
```

5.10 L'implementazione Myfaces.

Abbiamo già accennato al fatto che la prima release del framework JSF è stata definita nel Marzo del 2004. In realtà Java Server Faces rappresenta uno standard del quale ne esistono implementazioni differenti, tra le quali bisogna menzionare JSF RI (Reference Implementation) della Sun Microsystems e Myfaces di Apache.

In realtà ce ne sarebbero anche delle altre ma queste 2 sono sicuramente le più popolari.

Molti sviluppatori preferiscono la RI, perché è l'implementazione ufficiale della SUN, ma MyFaces ha alcune estensioni interessanti.

In realtà si possono usare le estensioni di MyFaces insieme alle JSF RI della SUN, se uno vuole.

Per la nostra applicazione abbiamo deciso di puntare sulla RI della Sun, ma siamo ricorsi alla release di Myfaces per usufruirne delle potenzialità dove la RI risultava carente.

In particolar modo JSF non prevede ancora uno standard per l'upload di un file, progetto portato avanti da Myfaces. Abbiamo sfruttato dell'implementazione di Myfaces anche l'inserimento della data attraverso un calendario in java script, funzionalità che ci è risultata utile nella fase di registrazione dell'utente.

5.10.1 Upload di un file

I web browsers forniscono una semplice via per inviare files a web application, ma le corrente versioni degli standard java per il web(servlets,JSP e JSF) non offrono alcun aiuto.

Il termine upload è qualcosa di sovrautilizzato. Un webmaster direbbe che egli "uploada" un file quando lo pubblica sul suo sito web. Uno sviluppatore web direbbe che egli implementa l'upload di un file quando crea un form HTML e uno script che permetta agli utenti di inviare files usando i loro web browsers.

Configurare JSF e le estensioni di MyFaces

Ciò che bisogna fare ed è ciò che è stato fatto durante il nostro progetto, è aggiungere il file myfaces-extensions.jar ai files JAR della JSF RI e commons-fileupload-1.0.jar nella directory WEB-INF/lib dell'applicazione.

In una tipica applicazione, oltre al tag dell'upload, bisogna configurare l'ExtensionsFilter nel descrittore web.xml della nostra applicazione in maniera tale che intercetti le richieste http prima della FacesServlet delle JSF.

Ecco la parte del web.xml che definisce il filtro e le sue proprietà, modificabili a nostro piacimento e per le nostre esigenze.

```
<filter>
    <filter-name>ExtensionsFilter</filter-name>
    <filter-class>
      org.apache.myfaces.component.html.util.ExtensionsFilter
    </filter-class>
    <init-param>
      <param-name>uploadMaxFileSize</param-name>
      <param-value>10m</param-value>
    </init-param>
    <init-param>
      <param-name>uploadThresholdSize</param-name>
      <param-value>100k</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>ExtensionsFilter</filter-name>
    <servlet-name>FacesServlet</servlet-name>
  </filter-mapping>
```

I 2 parametri del filtro dicono a Myfaces di tenere i files di dimensione minore dei 100K in memoria e ignorare I files che richiedano più di 10 MB di spazio sul disco.

Per usare JSF e MyFaces in una pagina web bisogna dichiarare le loro librerie di tag con la direttiva

```
<%@ taglib uri="http://myfaces.apache.org/extensions" prefix="x"%>
```

Il tag <h:form> non ha un attributo metodo, poichè supporta solo il metodo POST, ma ha un attributo enctype che devi usare se vuoi uploadare files, specificando il tipo multipart encoding per il form dei dati.

```
<f:view>

  <h:form id="MyForm" enctype="multipart/form-data" >
    ...
    <x:inputFileUpload id="myFileId"
      value="#{myBean.myFile}"
      storage="file"
      required="true"/>
    ...
  </h:form>

</f:view>
```

Ci sono molti casi nei quali l'utente ha bisogno di fare upload di files attraverso il loro browser, ma non esiste un modo ideale per gestire questi files sul lato server.

Tenere il contenuto del file in memoria è accettabile solo per piccoli files, mentre memorizzare il contenuto uploadato in file temporanei complica la situazione.

Myfaces ci lascia scegliere la soluzione appropriata per la nostra applicazione, ma questo frameworks ha alcuni problemi minori.

Non ci permette di eliminare i files temporanei quando non servono più, i nomi dei file a volte sono paths e non ci sono warnings quando l'utente cerca di uploadare files che sono troppo grandi.

Questi problemi possono essere risolti, essendo il codice sorgente disponibile.

5.10.2 Il calendario

Usare il calendario predefinito di Myfaces ha comportato le seguenti 2 operazioni:

```
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t" %>
```

per includere l'apposita libreria e

```
<t:inputDate value="#{user.data}"  popupCalendar="true" >
```

per caricarlo nella pagina.

5.11 L'applet Vimida.

A corredo dell'applicazione, già completa e funzionale nella sua struttura, abbiamo deciso di usufruire delle potenzialità di un applet disponibile gratuitamente in rete, l'applet Vimida.

Un'applet non è altro che una applicazione Java che gira su web. L'applet presenta qualche differenza con le applicazioni, infatti essa non ha nessun main ed è costituita da classi, chiamate come il file che le contiene, che estendono la classe Applet del package java.applet.

Vimida è quindi un'applet java per la visualizzazione di dati prodotti da analisi con Microarray, grazie all'uso della PCA(analisi dei componenti principali) e dell'algoritmo elmap (Elastic Map).

La *principal component analisis* (PCA) ha lo scopo di ridurre la complessità della rappresentazione dei geni. Supponiamo di avere la matrice delle espressioni dei geni, e che le espressioni di alcuni di essi siano correlate, la PCA ignora gli esperimenti ridondanti e fornisce una media pesata di alcuni esperimenti così da fornire un andamento dei dati più interpretabile. I componenti possono essere pensati come assi in uno spazio n-dimensionale, dove n è il numero di componenti, ogni asse rappresenta un andamento differente nei dati.

Dal punto di vista dell'interfaccia utente, viene aperta una finestra pop-up all'interno della quale viene caricata l'applet.

Da qui, l'utente ha la possibilità di interagire con quest'ultima modificando il tipo di analisi da effettuare sui dati (Linear PCA o Elastic Map), decidendo o meno se etichettare tutti i geni caricati nel file dei dati, annullare le scelte precedenti o visualizzare informazioni sul singolo gene selezionato nella finestra. La costruzione delle mappe può richiedere del tempo in quanto l'algoritmo deve compiere molte epoche di addestramento per dare il risultato all'utente.

Alcune delle possibilità offerte solo inoltre la visualizzazione della densità relativa di una classe e quella media, così come quella di analizzare una tabella di dati "trasposta" rispetto a quella originale.

Poiché il formato dei dati richiesto dall'applet risulta leggermente diverso da quello previsto nella nostra applicazione per la lettura dei files, è stato necessario realizzare delle apposite routines per la conversione dei formati dei file memorizzati nel database in formati leggibili dall'applet.

Da un punto di vista software l'applet è richiamata in una pagina della nostra applicazione tramite

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
 <head>
  <title>
    HTML Test Page
  </title>
 </head>
 <body>
  <br>
  <center>
   <applet  code="vdaoengine.TestApplet.class"  width=750 height=750>
    <param name=bgcolor value="000000">
    <param name=fgcolor1 value="ff0000">
    <param name=fgcolor2 value="ff00ff">
    <param name="datfile" value ="file:///c:/eclipse/temp.txt">
    <param name="settings_file" value = "file:///c:/eclipse/elmap.ini">
    <param name="microarray_type" value = "expression_real_values ">
    <param name="archive" value = "VDAOEngine.jar">
   </applet>
  </center>
 </body>
</html>
```

dove è possibile passare vari parametri quali il file dei dati e quello dell'addestramento.

5.12 Commons Email.

Ultimo aspetto che rimane da trattare è la gestione della funzionalità di invio email da parte dell'applicazione.

Un utente, per vari motivi (dimenticanza o smarrimento) potrebbe aver bisogno di recuperare i dati di accesso all'applicazione.

Ciò può essere ottenuto attraverso l'invio di un'email da parte del sistema contenente i dati di interesse per l'utente.

Per realizzare questa funzionalità abbiamo usufruito delle Commons-Email, un progetto di Jakarta, il cui scopo è quello di fornire delle API per l'invio delle email.

Il servizio è costruito sulla base delle API Java Mail e nasce con lo scopo di semplificarne l'uso.

Per usufruire del servizio è necessario quindi che l'ambiente sia JavaMail ready, cioè con le librerie *activation.jar* e *mail.jar* nel classpath.

Fatto questo è possibile aggiungere la libreria *commons-email-1.0* che fornisce il servizio.

Per le Java Mail abbiamo usato la release 1.4.

Le principali classi fornite dal progetto sono:

- **SimpleEmail** – Questa classe è usata per inviare email di semplice testo.
- **MultiPartEmail** – Questa classe è usata per inviare messaggi con l'aggiunta di allegati.

Per il nostro semplice scopo è bastato usare SimpleEmail, i cui unici parametri da settare sono risultati:

```
SimpleEmail email = new SimpleEmail();

email.setHostName("out.alice.it");
email.addTo(mail, "Daniele Battinelli");
email.setFrom("gbattine@alice.it", "Giuseppe
      Battinelli");
email.setSubject("Recupero dati applicazione
      microarray");
email.setMsg("Username="+username+" ,
      Password="+password);
email.send();
```

CONCLUSIONI

Ed eccoci giunti alla fine del lavoro.

Finalmente possiamo stilare un resoconto del lavoro svolto, analizzare quelli che erano gli obiettivi prefissati e stabilire quanti di questi , e con che grado di soddisfazione, siano stati raggiunti.

Possiamo dire che quanto ci eravamo prefissati è stato raggiunto in maniera soddisfacente, in quanto l'applicazione è perfettamente funzionante e tutte le richieste pervenute sono state soddisfatte.

Inoltre è stata aggiunta, oltre le richieste iniziali, l'integrazione dell'applet Vimida con la conseguente possibilità di visualizzazione grafica dei dati normalizzati dell'esperimento.

Sono state effettuate numerose sessioni di testing al fine di validare l'applicazione e scoprire eventuali malfunzionamenti, per cui l'applicazione ha subito nella fase finale alcuni aggiornamenti atti ad eliminare problemi emersi nell'uso da parte degli addetti.

Per quanto riguarda gli sviluppi futuri è possibile dire che, allo stato attuale, nel database, sono presenti esclusivamente i dati relativi alle piattaforme Illumina, Agilent, Amershaw e Affymetrix. Pertanto potranno essere introdotti in seguito i dati relativi ad altre piattaforme di interesse.

Tale operazione è facilmente eseguibile dall'Amministratore del Sistema.

Altro possibile sviluppo futuro riguarda l'aggiunta di testi e messaggi in altre lingue.

Anche questo aspetto è facilmente realizzabile tramite l'utilizzo di altri resource bundle, il tutto senza dover modificare nulla all'interno dell'applicazione.

Lo spunto più interessante sarà però l'analisi dei dati degli esperimenti. L'applicazione al giorno d'oggi si "limita" ad analizzare la parte dell'esperimento nel suo complesso, dando la possibilità all'utente di eseguire ricerche anche sui files, ma non all'interno dei dati numerici dei singoli files.

Sarà necessario definire particolari indici, anche complessi, per permettere in futuro anche query sui dati "numerici" degli esperimenti.

In realtà sono state già progettate ed incluse nell'applicazione delle routines per la corretta lettura di un file di esperimenti, per cui la soluzione di questo futuro problema potrà partire da queste basi già presenti nell'applicazione.

Inoltre potrebbe essere migliorata, in futuro, l'operazione di lettura e memorizzazione dei files dei dati, migliorando e ottimizzando le procedure che consentono l'invio del file nel database sotto forma di un array di byte.

Migliorando queste procedure sarà possibile incrementare anche le prestazioni del sistema.

Il punto da sottolineare è che questa applicazione, pur nel suo valore e nella sua funzionalità, non deve assolutamente essere considerata come un punto di arrivo, bensì un punto di partenza per lo sviluppo di nuove potenzialità da aggiungere che possano facilitare e rendere più completa l'analisi di questi dati prodotti dai microarray ed essere d'ausilio agli addetti ai lavori.

Nel corso della realizzazione dell'applicazione, è stato presentato il framework Jsf, cercando di illustrare le caratteristiche basilari e quelle ritenute più utili per chi opera nello sviluppo di software per il Web.

Nella letteratura specializzata, nei forum e nei vari siti consultati, Jsf viene presentato come il grande antagonista del framework che finora ha governato la scena, Struts. Ci sono molti presupposti, come detto nei capitoli precedenti, per cui Jsf diventi nei prossimi l'anni l'erede di Struts o il suo più naturale completamento.

Le discussioni al riguardo sono moltissime e ciò inizialmente aveva creato molta curiosità e forse è stato proprio questo il motivo che ci ha portato alla sua scelta. Abbiamo deciso di puntare su un framework "giovane" ma con delle potenzialità enormi grazie anche alla sua semplice interoperabilità con altri frameworks ed estendibilità.

Per quanto riguarda il software prodotto, possiamo dire che la sua manutenzione è facilmente eseguibile dal solo menu di amministrazione nel quale sono state predisposte tutte le funzioni necessarie ad una corretta inizializzazione e gestione dei dati dell'applicazione da parte dell'amministratore.

APPENDICE A

SRS dell'applicazione MicroArray Web secondo lo standard IEEE Std 830-1998

1. Introduzione

1.1. Propositi.

Si vuole progettare un sistema software di ausilio nella gestione di un archivio di esperimenti genetici basati sulla tecnica del MicroArray. Questo documento ha lo scopo di definire i requisiti e le specifiche del prodotto software in esame, al fine di facilitarne la realizzazione e la validazione.

1.2. Obiettivi.

Il sistema deve mantenere un archivio degli utenti che possono accedervi (compresi i relativi privilegi) e gestire l'archiviazione di un esperimento attraverso tutte le sue fasi e la sua consultazione secondo utili e funzionali "percorsi di navigazione".

Inoltre deve fornire ulteriori servizi differenziati a seconda del ruolo dei possibili utenti del sistema, siano essi di tipo utente o amministratore.

Tutte le funzionalità del sistema sono usufruibili esclusivamente da utenti registrati, per gli utenti non registrati o visitatori l'unica funzionalità prevista è la registrazione.

1.3.Definizioni,acronimi e abbreviazioni.

- Password: insieme di caratteri alfanumerici che abilitano l'accesso ad aree riservate a utenti registrati.
- Username: insieme di caratteri alfanumerici per l'identificazione di un utente.
- Gruppo di ricerca: insieme di caratteri alfanumerici che identificano il gruppo di ricerca di appartenenza di un utente, utile al fine di un'identificazione univoca dello stesso.

- Utenti del sistema informativo:sono tutti coloro che possono accedere alle funzionalità dell'applicazione(naturalmente ve ne sono diverse categorie con diritti differenti).
- Utenti registrati: sono gli utenti che sono già stati precedentemente registrati nell'archivio dell'applicazione e che quindi sono possessori di un account che li identifica in modo univoco ed hanno quindi accesso alle funzionalità del sistema.
- Account utente: associato ad ogni utente registrato è descritto da tre parametri, la username, la password e il gruppo di appartenenza.
- Utenti generici: sono utenti che, non essendo mai stati registrati, non possono essere oggetto di controllo da parte dell'applicazione, quindi è loro consentito un tipo di accesso molto ristretto che consente di ottenere soltanto la registrazione tramite un apposito form.
- Esperimento: in questo documento, con questo termine si fa riferimento a qualsiasi esperimento catalogato, sia esso eseguito su una delle qualsiasi piattaforme a disposizione.
- Piattaforma: con questo acronimo si intende far riferimento allo specifico array sul quale è stato eseguito l'esperimento. La nostra applicazione prevederà l'uso di 4 piattaforme di partenza, l'Amershaw, l'Illumina, l'Agilent e l'Affymetrix, con la possibilità da parte dell'amministratore, di aggiungerne di ulteriori in futuro.
- Organismo: il gene e le specie e sottospecie dell'organismo dal quale è derivato il biomateriale.
- Visibilità esperimento: descrive l'ambito di visibilità di un esperimento, fondamentale nella selezione dei giusti esperimenti da mostrare in fase di ricerca all'utente. Può prevedere 3 valori, privato, gruppo e pubblico.

- Per visibilità privata si intende che quell'esperimento potrà essere visibile solo all'utente che lo ha inserito.
- Per visibilità gruppo si intende che quell'esperimento potrà essere visibile a tutti gli utenti dell'applicazione appartenenti ad uno stesso gruppo.
- Per visibilità pubblica si intende che quell'esperimento potrà essere visibile a tutti gli utenti registrati del sistema.
- Utente interdetto: è un utente al quale l'amministratore ha bloccato l'accesso all'applicazione a causa di un errato inserimento dei dati di login per 3 volte di seguito o per altri motivi amministrativi.
- Utente in attesa di conferma: quando un utente si registra non ha immediato accesso all'applicazione, ma deve attendere una conferma da parte dell'amministratore che gli garantisca l'accesso.
- Utente loggato: si intende un utente che al momento è in navigazione nelle pagine dell'applicazione.
- Amplificazione: Metodo usato per amplificare l'acido nucleico estratto.
- Nome array: Nome dato ad una particolare "versione" dell'array per differenziarlo da un altro array dello stesso tipo.
- Dimensioni array: Le dimensioni fisiche dell'array, date per righe e colonne di quest'ultimo.
- Autore, laboratorio e contatti: persone, nomi di organizzazioni e dettagli(indirizzo, telefono, fax, email, url) degli autori dell'esperimento.
- Descrizione esperimento: testo libero di descrizione dell'esperimento e link a possibili pubblicazioni elettroniche.

- Tipo di esperimento: è un attributo che specifica la tipologia dell'esperimento.
- Fattori Sperimentali: parametri o condizioni testate nell' l'esperimento.
- Metodo di estrazione: il protocollo usato per estrarre gli acidi nucleici dal campione.
- Numero di features: numero di reporters sull'array.
- Protocollo di ibridazione: documentazione della serie di passaggi eseguiti nell'ibridazione, inclusi: soluzione, agente bloccante e concentrazione usati; tempo; concentrazione; volume; temperature e descrizione degli strumenti di ibridazione.
- Software Analisi: documentazione del set di passaggi eseguiti per quantificare l'immagine inclusa, il software di analisi, l'algoritmo e tutti i parametri usati.
- Label: protocollo di marcatura usata.
- Numero di ibridazioni: numero di ibridazioni eseguite nell'esperimento.
- Organismo: Il gene e le specie (e sottospecie) dell'organismo dal quale è derivato il biomateriale.
- Tipo di piattaforma: tipo di tecnologia usata per fissure la sequenza biologica sull'array.
- Controlli di qualità: Misure eseguite per garantire o misurare la qualità: replicati(numero e descrizione) o altre.
- Campione: il materiale biologico dal quale sono stati estratti gli acidi nucleici for una successive marcatura e ibridazione. Miame distingue tra origine del campione (biosource), il suo trattamento, la preparazione dell'estratto e la sua marcatura.
- Protocollo di scansione: documentazione del set di passaggi eseguiti per scannerizzare l'array e generare un immagine, includendo una descrizione degli strumenti di scannerizzazione e dei settaggi dei parametri.

- Tipo di trattamento: tipo di manipolazione applicata al biomateriale con gli obiettivi di generare una delle variabili sotto studio.
- Analisi effettuate: rappresenta il numero di analisi statistiche effettuate su un singolo esperimento.
- Campione di prova: è il campione di prova considerato nell'incrocio dei campioni per la descrizione delle analisi statistiche effettuate sull'esperimento.
- Campione di riferimento: è il campione test che viene confrontato con il campione di prova.
- Dati grezzi: i dati prodotti dalla scansione delle immagini dell'array.
- Dati Normalizzati: i dati ottenuti dalle procedure di normalizzazione.
- GeneList: le liste dei geni regolati, in base a test e parametri statistici.
- Tabella Annotazioni: tabella fornita dal produttore dell'array.

1.4.Riferimenti.

Il software è completamente nuovo quindi non vi sono referenze degne di nota.

1.5.Generalità.

L'intento di questo documento è quello di descrivere le funzionalità che il software deve soddisfare, le quali saranno specificate nel capitolo successivo in modo chiaro e conciso.

2. Descrizione generale

2.1 Prospettiva dell'applicazione

Questo prodotto non si integra con nessun altro sistema software ed è indipendente ed autonomo per quanto riguarda l'elaborazione (stand-alone).

MicroArrayWeb necessita della presenza di un DBMS come indicato in 2.1.6 ed è indipendente da particolari sistemi hardware.

2.1.1 Interfaccia utente

L'interfaccia utente è quella classica delle applicazioni Windows. In particolare all'avvio di MicroArrayWeb viene mostrata una pagina tramite la quale l'utente può effettuare l'autenticazione.

Se l'utente si autentica viene mostrata una pagina di menu, con la possibilità di accedere alle funzionalità dell'applicazione tramite una serie di possibilità personalizzate in base alla tipologia di utente che ha effettuato l'accesso.

2.1.2 Interfaccia Hardware

MicroArrayWeb non prevede l'uso di dispositivi Hardware esterni alla configurazione di sistema riportata nella successiva sezione dei Vincoli di Progetto.

2.1.3 Interfaccia Software

Il prodotto Software per funzionare necessita di:

Sistema Operativo Windows 98,ME oppure XP (www.microsoft.com).

2.1.4 Interfaccia di Comunicazione

La comunicazione fra il sistema software "lato client" e quello "lato server", si svolge utilizzando la rete Internet e si basa fondamentalmente sull'accesso ad un database esterno al server stesso.

2.1.5 Operazioni

MicroArrayWeb prevede che tutte le sue funzionalità siano avviate da uno dei suoi utenti che deve essersi preventivamente autenticato e che abbia diritto d'accesso a quella funzionalità.

2.1.6 Database

L'applicazione per funzionare necessita di un database. Per motivi pratici e di sicurezza si prevede che questo database sia creato e gestito esternamente a MicroArrayWeb, in particolare alcuni archivi devono essere popolati dall'amministratore all'atto dell'installazione e prima dell'utilizzo da parte dell'utente.

2.1.7 Adattamento all'ambiente

MicroArrayWeb prevede all'atto dell'installazione la memorizzazione dei dati relativi alle piattaforme in uso, agli organismi catalogati, ai gruppi di ricerca in archivio e ad altri parametri del sistema.

2.2 Funzionalità

MicroArrayWeb deve fornire le funzionalità di seguito riportate che,per motivi di chiarezza, sono state organizzate in categorie:

a) Categoria Autenticazione – MicroArrayWeb provvede all'autenticazione dell'utente consentendogli di accedere solo a quelle funzionalità che gli competono sulla base dell'account ad esso associato. L'obiettivo è fare in modo che l'accesso al sistema non sia consentito a chiunque (registrato o meno), ma che si abbia un accesso "protetto".

Allo stesso tempo però si deve fare in modo che ciascun utente possa compiere solo le operazioni che gli sono consentite.

Nel momento in cui un utente necessiterà di addentrarsi nell'applicazione per usufruirne dei servizi dovrà registrarsi, se non lo ha già fatto in passato.

L'applicazione deve inoltre concedere sia all'amministratore che all'utente la possibilità di effettuare il logout, ovvero di abbandonare l'applicazione liberando le risorse occupate.

b) Categoria utenti - Gestione Anagrafica utenti.

Tale archivio conterrà le informazioni necessarie sugli utenti per il controllo degli accessi alle risorse dell'applicazione e per la gestione di questi ultimi da parte dell'amministratore.

L'applicazione dovrà conferire al riguardo servizi differenziati in base al ruolo dell'utente:

- all'amministratore deve consentire la possibilità di interdire un utente, sbloccarlo, eliminarlo, renderlo attivo dopo la registrazione e visualizzarne i relativi esperimenti. Inoltre deve consentire all'amministratore di promuovere un utente registrato ad amministratore.
- ad un utente esterno dovrà fornire la possibilità di registrarsi, dopodichè l'utente passerà nello stato di attesa di conferma da parte dell'amministratore.
- ad utente registrato e all'amministratore la possibilità di consultare i suoi dati e di ricevere un'email con il suo account in caso di dimenticanza dei dati di registrazione.

c) Categoria esperimenti - Gestione dei dati degli esperimenti.

L'archivio degli esperimenti serve per soddisfare al meglio le richieste degli utilizzatori. Di ogni esperimento devono essere descritti i passi affrontati e tutte le misurazioni effettuate, al fine di consentire all'utente in un secondo momento di "ricostruire" l'intero processo che ha portato a determinare i risultati di un particolare esperimento.

Questa categoria comprende delle funzionalità specifiche da compiere sull'archivio esperimenti.

Ad un utente registrato deve essere consentito fare:

- Ricerca su ogni tipo di informazione fondamentale per l'esperimento.
- Deve essere data la possibilità di effettuare ricerche su tutti i fattori fondamentali dell'esperimento, a partire da qualsiasi informazione disponibile .

In particolare abbiamo previsto 5 percorsi di ricerca diversi,attraverso i quali reperire le informazioni di interesse per l'utente:

- Ricerca di un esperimento sulla base della visibilità di quest'ultimo
- Ricerca di un esperimento sulla base di 5 parametri selezionabili da parte dell'utente,cioè:
  - Fattore sperimentale
  - Autore
  - Laboratorio in cui è stato svolto l'esperimento
  - Tipologia dell'esperimento
  - Organismo usato nel trattamento dei campioni dell'esperimento.
- Visualizzazione dei risultati della ricerca.

  Deve essere data la possibilità di visualizzare in maniera funzionale e chiara i vari risultati richiesti nella fase di ricerca.
- Download dei dati dell'esperimento.

Deve essere data la possibilità, individuato un particolare esperimento dai meccanismi di ricerca, di scaricarne i file contenenti i risultati di quest'ultimo.

- Inserimento dati esperimento.

  Deve essere data la possibilità all'utente di inserire in maniera chiara e funzionale un esperimento nell'archivio attraverso tutte le fasi che lo compongono, partendo dai dettagli sperimentali, passando per l'inserimento delle informazioni sui campioni fino a giungere ai files che descrivono i risultati dello stesso.

- Modifica dati esperimento.

  Deve essere data la possibilità all'utente di modificare i dati di un esperimento da lui inserito, sia per quanto riguarda le informazioni di realizzazione dell'esperimento che i files riguardanti i risultati dell'esperimento stesso.

- Cancellazione dati esperimento.

  Deve essere data la possibilità all'utente di cancellare un esperimento da lui inserito.

- Visualizzazione grafica dei dati normalizzati.

  Deve essere data la possibilità all'utente di visualizzare graficamente, una volta selezionato un'esperimento, i dati normalizzati dello stesso secondo le tecniche dell'intelligenza artificiale.

d) Categoria amministrazione - Gestione dei dati riguardanti l'applicazione.

Questa categoria è dal nome dedicata esclusivamente all'amministratore del sistema e comprende varie funzionalità:

1) Aggiunta nuove piattaforme : l'applicazione deve fornire all'amministratore la possibilità di inserire una scheda contenente tutti i valori relativi ad una nuova piattaforma nel sistema.

2) Aggiunta nome piattaforma : l'applicazione deve fornire all'amministratore la possibilità di inserire nuovi nomi di piattaforme utilizzabili nel sistema.

3) Aggiunta tipologia piattaforma : l'applicazione deve fornire all'amministratore la possibilità di aggiungere nuove tipologie di piattaforme utilizzabili nel sistema.

4) Modifica valori standard delle varie piattaforme: l'applicazione deve fornire all'amministratore la possibilità di modificare i valori standard delle quattro piattaforme considerate nei vari esperimenti.

5) Eliminazione piattaforme : l'applicazione deve fornire all'amministratore la possibilità di eliminare dal sistema i dati relativi a piattaforme non più in uso.

6) Aggiunta di nuovi gruppi di ricerca: l'applicazione deve fornire all'amministratore la possibilità di aggiungere dei nuovi gruppi di ricerca disponibili per l'inserimento nell'archivio di nuovi utenti.

7) Eliminazione di un gruppo di ricerca: l'applicazione deve fornire all'amministratore la possibilità di eliminare gruppi di ricerca non più disponibili per l'archiviazione di nuovi utenti, operazione che include l'eliminazione di tutti gli utenti facenti parte del gruppo di ricerca.

8) Aggiunta di un nuovo organismo: l'applicazione deve fornire all'amministratore la possibilità di aggiungere dei nuovi organismi selezionabili dall'utente nella fase di inserimento dei dati di un esperimento.

9) Eliminazione di un organismo: l'applicazione deve fornire all'amministratore la possibilità di eliminare degli organismi non più utilizzati nello svolgimento degli esperimenti e quindi non più di interesse per l'applicazione.

10) Aggiunta di un test statistico: l'applicazione deve fornire all'amministratore la possibilità di aggiungere un nuovo test statistico utilizzabile nella fase di inserimento dell'esperimento.

11) Eliminazione di un test statistico: l'applicazione deve fornire all'amministratore la possibilità di eliminare un test

statistico non più utilizzabile nella fase di inserimento dell'esperimento.

12) Aggiunta di un parametro statistico: l'applicazione deve fornire all'amministratore la possibilità di aggiungere un nuovo parametro statistico utilizzabile nella fase di inserimento dell'esperimento.

13) Eliminazione di un parametro statistico: l'applicazione deve fornire all'amministratore la possibilità di eliminare un parametro statistico non più utilizzabile nella fase di inserimento dell'esperimento.

14) Aggiunta di un laboratorio: l'applicazione deve fornire all'amministratore la possibilità di aggiungere un nuovo laboratorio utilizzabile nella fase di inserimento dell'esperimento.

15) Eliminazione di un laboratorio: l'applicazione deve fornire all'amministratore la possibilità di eliminare un laboratorio non più utilizzabile nella fase di inserimento dell'esperimento.

16) Aggiunta di un fattore di correzione: l'applicazione deve fornire all'amministratore la possibilità di aggiungere un nuovo fattore di correzione utilizzabile nella fase di inserimento dell'esperimento.

17) Eliminazione di un fattore di correzione: l'applicazione deve fornire all'amministratore la possibilità di eliminare un fattore di correzione non più utilizzabile nella fase di inserimento dell'esperimento.

18) Aggiunta di una nuova tipologia di esperimento: l'applicazione deve fornire all'amministratore la possibilità di aggiungere una nuova tipologia di esperimento utilizzabile nella fase di inserimento di quest'ultimo.

19) Eliminazione di una tipologia di esperimento: l'applicazione deve fornire all'amministratore la possibilità di eliminare una tipologia di esperimento non più utilizzabile nella fase di inserimento dell'esperimento.

20) Visualizza dati utenti: l'applicazione deve fornire all'amministratore la possibilità di visualizzare i dati anagrafici relativi a tutti gli utenti del sistema, con la possibilità di visualizzare gli esperimenti realizzati dall'utente selezionato..

21) Visualizza utente : l'applicazione deve fornire all'amministratore la possibilità di ricercare la scheda di un utente a partire dalla sua username.

22) Modifica dati utente: l'applicazione deve fornire all'amministratore la possibilità di modificare i dati di un utente selezionato attraverso la sua username..

23) Eliminazione utente : l'applicazione deve fornire all'amministratore la possibilità di eliminare dal sistema i dati relativi ad un utente non più facente parte dell'applicazione, con il conseguente settaggio della username relativa agli esperimenti di questi ultimi all'username dell'amministratore, per garantire la possibilità all'amministratore in futuro di consultare un esperimento di un utente non più presente nel sistema.

24) Interdizione utente : l'applicazione deve fornire all'amministratore la possibilità di interdire dal sistema un utente .

25) Sblocco utente : l'applicazione deve fornire all'amministratore la possibilità di sbloccare un utente da una precedente interdizione.

26) Conferma utente : l'applicazione deve fornire all'amministratore la possibilità di dare accesso all'applicazione ad un utente precedentemente registrato e non ancora confermato dall'amministratore.

27) Promozione utente ad amministratore: l'applicazione deve fornire all'amministratore la possibilità di promuovere un utente precedentemente registrato ad amministratore del sistema.

2.3 Caratteristiche degli utenti.

L'applicazione MicroArrayWeb è destinata ad utenti che abbiano esperienza di utilizzo di Personal Computer e relativi software per la produttività personale.

Si presuppone che l'utente sappia navigare in Internet, sappia compilare un Form e possieda un recapito di posta elettronica; si suppone inoltre che conosca la modalità di inserimento di un esperimento che avviene attraverso una parte iniziale di tipo "descrittiva" ed una parte finale riguardante i veri e propri "risultati" dell'esperimento.

All'interno degli utenti che possono fruire delle funzionalità dell'applicazione possiamo fare una classificazione in base al ruolo e alle funzionalità disponibili.

Visitatore:

Al fine di garantire l'integrità e la protezione dei dati l'accesso dell'utente generico è particolarmente ristretto. È consentita soltanto la consultazione della pagina iniziale, nel caso in cui l'utente voglia usufruire dei servizi dell'applicazione deve registrarsi e compilare il form relativo alla richiesta di iscrizione.

Utente registrato:

Quando il visitatore si registra, ottenendo un account, diviene un utente dell'applicazione. Grazie alla registrazione l'utente ottiene privilegi superiori rispetto a quelli di un visitatore.

Con l'account ottenuto potrà infatti non limitarsi a consultare la pagina iniziale, ma potrà accedere alle funzionalità offerte dall'applicazione.

Amministratore.

Gode di tutti i privilegi degli utenti registrati, in più gode di privilegi supplementari quali quelli di poter aggiornare l'archivio degli utenti, di poter interdire e sbloccare un utente e di poter aggiornare i dati delle piattaforme.

2.4 Vincoli

L'applicazione richiede il settaggio dell'heap size di Tomcat a 256 Mb di memoria minima,contro i 64 di default, a causa della grossa mole dei file che vengono elaborati dall'applicazione.

2.5 Assunzioni e dipendenze

Web container Apache Tomcat con supporto per Jsf e per database Mysql.

2.6 Dilazione di requisiti

Possibilità di indicizzazione per effettuare queries sui dati stessi.

3. Requisiti Specifici.

3.1 Interfacce esterne

Non ci sono aggiunte significative da fare a quanto detto nella parte 2.1.

3.2 Requisiti funzionali

1) Classe Utente/Amministratore

3.2.1 Recupero dati registrazione.

3.2.1.1 Introduzione

Funzione che permette all'utente/amministratore la richiesta di invio, tramite email, dei dati di login inseriti in fase di registrazione in caso di smarrimento di questi ultimi.

3.2.1.2 Input

L'utente/amministratore inserisce in 2 form separati l'indirizzo email.

Dati

- Email (obbligatorio).
- E-mail verifica (obbligatorio).

3.2.1.3 Funzione

- Controlla che entrambi i form siano stati riempiti.
- Controlla la validità del formato dei dati.
- Controlla che i valori inseriti nei 2 campi siano uguali.
- Controlla che esista un utente registrato con quell'indirizzo e-mail.
- Conferma l'avvenuta memorizzazione dei dati e l'immediato invio dei dati personali.

3.2.1.4 Output

Form per la digitazione dell'indirizzo e-mail.

Messaggi

"Il campo relativo all'e-mail non può essere vuoto".

"Il formato dell'e-mail non è valido".

"I due valori di e-mail inseriti non coincidono".

"L'e-mail non è presente nel database degli utenti registrati".

"Mail inviata con successo!".

3.2.2 Login

3.2.2.1 Introduzione

Questa funzione permette all'utente o all'amministratore di accedere al sistema.

3.2.2.2 Input

Dati

- Username (obbligatorio)
- Password (obbligatorio)
- Gruppo di ricerca (obbligatorio)

3.2.2.3 Funzione

Visualizzazione di un form per l'input dei dati, sui quali il sistema esegue le seguenti operazioni :

- Controlla che i dati obbligatori siano stati digitati.
- Verifica che username e gruppo di ricerca corrispondano ad un utente registrato o ad un amministratore, che abbiano associato ad essi un contatore "tentativi a disposizione">0;
- Controlla che la password inserita sia quella presente nel db.
- Controlla che l'utente non sia già loggato, interdetto o in attesa di conferma da parte dell'amministratore.

- Nel caso di successo di login registra i dati di accesso.
- Verifica i diritti di accesso.
- Visualizza un menu dei servizi differenziato a seconda del tipo utente/amministratore.

.

3.2.2.4 Output

Menù di accesso alle aree di utente.

Dati relativi all'accesso registrati in archivio :

- Username e nome gruppo.
- Numero tentativi di accesso.

Messaggi

"Username e/o Password errati".

"Utente già in navigazione"

"Utente interdetto"

"Utente in attesa di conferma da parte dell'amministratore"

"Utente non registrato, seguire la procedura di registrazione"

"Password sbagliata,hai altri 2 tentativi"

"Password sbagliata, hai un altro tentativo"

"Hai sbagliato 3 volte la password,utente interdetto,contattare l'amministratore".

3.2.3 Ricerca esperimenti nell'archivio per ambito di visibilità.

3.2.3.1 Introduzione

Funzione che permette all'utente/amministratore la visualizzazione dei dati relativi agli esperimenti selezionati, sulla base della scelta tra ambito di visibilità privato, gruppo o pubblico.

3.2.3.2 Input

Esistono 3 tipi di ricerca possibili nell'applicazione:

- Ambito di visibilità dell'esperimento (obbligatorio).

3.2.3.3 Funzione

Visualizza un Form per la ricerca di esperimenti relativi all'ambito di visibilità inserito dall'utente/amministratore.

- Controlla che almeno un parametro sia stato digitato.

3.2.3.4 Output

Form con parametri da scegliere.

Dati relativi alla richiesta fatta.

Messaggi

"Nessun ambito di visibilità inserito"

"Nessun esperimento trovato".

3.2.4 Ricerca esperimenti attraverso fattori inseriti dall'utente.

3.2.4.1 Introduzione

Funzione che permette all'utente/amministratore la visualizzazione dei dati relativi agli esperimenti selezionati attraverso la specifica di alcuni parametri a discrezione dell'utente.

3.2.4.2 Input

- Autore.
- Laboratorio.
- Fattore Sperimentale.
- Tipologia di esperimento.
- Organismo.

3.2.4.3 Funzione

- Visualizza gli esperimenti corrispondenti ai parametri inseriti.

3.2.4.4 Output

Dati degli esperimenti richiesti.

Messaggi

"Nessun esperimento trovato".

3.2.5 Ricerca esperimenti attraverso una parola chiave.

3.2.5.1 Introduzione

Funzione che permette all'utente/amministratore la visualizzazione dei dati relativi agli esperimenti selezionati attraverso la specifica di una parola chiave a discrezione dell'utente.

3.2.5.2 Input

- Testo libero

3.2.5.3 Funzione

- Visualizza gli esperimenti corrispondenti al testo inserito per il quale è stata trovata una corrispondenza nel campo descrizione dell'esperimento.

3.2.5.4 Output

Dati degli esperimenti richiesti.

Messaggi

"Nessun esperimento trovato".

3.2.6 Ricerca campioni attraverso un fattore sperimentale e il relativo valore.

3.2.6.1 Introduzione

Funzione che permette all'utente/amministratore la visualizzazione dei dati relativi ai campionoi selezionati attraverso la specifica di un fattore sperimentale e del relativo valore.

3.2.6.2 Input

- Fattore sperimentale (obbligatorio)
- Valore(obbligatorio).

3.2.6.3 Funzione

- Visualizza i campioni corrispondenti ai parametri inseriti.

3.2.6.4 Output

Dati i campioni richiesti.

Messaggi

"Nessun campione trovato".

3.2.7 Ricerca sulle analisi statistiche effettuate su un esperimento attraverso l'inserimento dei valori relativi a molteplici fattori.

3.2.7.1 Introduzione

Funzione che permette all'utente/amministratore la visualizzazione dei dati relativi alle analisi statistiche effettuate sugli esperimenti attraverso l'inserimento dei valori relativi a molteplici fattori caratterizzanti l'analisi stessa.

3.2.7.2 Input

- Campione di prova.
- Campione di riferimento.
- Test statistico.
- Parametro statistico .
- Valore del parametro statistico.

3.2.7.3 Funzione

- Visualizza le analisi effettuate sull'esperimento corrispondenti ai parametri inseriti.

3.2.7.4 Output

Dati relativi alle analisi fatte, con possibilità di scaricare per ogni analisi il file contentente le liste dei geni regolati.

Messaggi

"Nessuna analisi trovata".

3.2.8 Inserimento esperimenti.

3.2.8.1 Introduzione

Funzione che permette all'utente/amministratore l'archiviazione di un nuovo esperimento e dei relativi dati.

3.2.8.2 Input

L’inserimento avviene tramite più pagine di form che contengono tutti i parametri necessari per la descrizione esaustiva di un esperimento.

Dati

- Breve descrizione dell'esperimento (obbligatorio).
- Autore (obbligatorio).
- Laboratorio dell'esperimento(obbligatorio).
- Contatti.
- Links e url.
- Controlli di qualità.
- Numero di fattori sperimentali (obbligatorio).
- Numero di parametri statistici (obbligatorio).
- Nome della piattaforma (obbligatorio)
- Numero di ibridazioni.
- Numero di analisi statistiche effettuate sull’esperimento (obbligatorio).
- Spazio di visibilità dell'esperimento (obbligatorio).
- Protocollo di scansione usato.
- Software di analisi.
- Tipo Normalizzazione.
- Tipologia di esperimento (obbligatorio).
- Scala Logaritmica (obbligatorio).
- Test.
- Fattore di correzione (obbligatorio).
- Elenco fattori sperimentali (obbligatorio).
- Elenco parametri statistici (obbligatorio).
- Numero di campioni (obbligatorio).

3.2.8.3 Funzione

Visualizzazione di vari forms per l’input dei dati, sui quali il sistema esegue le seguenti operazioni :

- Controlla che i campi relativi alla Breve Descrizione dell'esperimento, all'Autore, al Laboratorio dell'esperimento, al Numero di fattori sperimentali, al numero di parametri statistici, al Nome della piattaforma, al numero di analisi statistiche, allo Spazio di visibilità dell'esperimento, alla Tipologia di esperimento, alla Scala logaritmica, al Fattore di correzione, all'Elenco dei fattori sperimentali, all'Elenco dei parametri statistici, al Numero di campioni non siano vuoti.
- Controlla la validità del formato specifico per alcuni campi.
- Registrazione dei dati di input in archivio.

3.2.8.4 Output

- Tutti i dati letti in input registrati in archivio.

  Messaggi

  "Inserimento effettuato con successo".

3.2.9 Inserimento campioni.

3.2.9.1 Introduzione

Funzione che permette all'utente/amministratore l'archiviazione dei dati relativi ai campioni esaminati durante l'esperimento.

3.2.9.2 Input

L'inserimento avviene tramite una pagina di form di tipo tabellare, dove ogni riga consente l'inserimento dei valori relativi ad un campione analizzato.

Dati

- NomeCampione (obbligatorio).
- Organismo (obbligatorio).
- Protocollo di trattamento

- Metodo di estrazione
- Protocollo di marcatura
- Ibridazione
- Tipo di Amplificazione

3.2.9.3 Funzione

- Visualizza un numero di righe di forms pari al numero di campioni esaminato nell'esperimento in esame.
- Controlla che i parametri obbligatori siano inseriti.
- Controlla la validità del formato per alcuni campi specifici.

3.2.9.4 Output

- Tutti i dati letti in input registrati in archivio.

Messaggi

"Inserimento effettuato con successo".

3.2.10 Inserimento risultati esperimenti.

3.2.10.1 Introduzione

Funzione che permette all'utente/amministratore l'archiviazione dei files riguardanti i risultati dell'esperimento considerato.

3.2.10.2 Input

L'inserimento avviene tramite una pagina di form di tipo tabellare, dove ogni riga consente l'inserimento di un particolare tipo di risultati.

Dati

- Dati grezzi (obbligatorio).
- Dati normalizzati (obbligatorio).
- Tabella annotazioni (obbligatorio).

3.2.10.3 Funzione

- Controlla che i parametri obbligatori siano inseriti.

- Controlla la validità del formato per alcuni campi specifici.

3.2.10.4 Output

- Tutti i dati letti in input registrati in archivio.

Messaggi

"Inserimento effettuato con successo".

3.2.11 Inserimento analisi statistiche sull'esperimento.

3.2.11.1 Introduzione

Funzione che permette all'utente/amministratore l'inserimento dei dati relativi alle analisi statistiche effettuate sull'esperimento.

3.2.11.2 Input

L'inserimento avviene tramite una pagina di form di tipo tabellare, dove ogni riga consente l'inserimento di una particolare analisi eseguita sull'esperimento.

Dati

- Campione di prova.
- Campione di riferimento.
- Test statistico.
- Parametro statistico .
- Valore del parametro statistico.
- File contenente la lista dei geni regolati relativa all'analisi.

3.2.11.3 Funzione

- Controlla che i parametri obbligatori siano inseriti.
- Controlla la validità del formato per alcuni campi specifici.

3.2.11.4 Output

- Tutti i dati letti in input registrati in archivio.

Messaggi

"Inserimento effettuato con successo".

3.2.12 Visualizzazione grafica dei dati normalizzati dell'esperimento.

3.2.12.1 Introduzione

Funzione che permette all'utente/amministratore la visualizzazione grafica, secondo tecniche dell'intelligenza artificiale, dei dati normalizzati dell'esperimento.

3.2.12.2 Input

L'apposito comando.

3.2.12.3 Funzione

- Carica il file dei normalizzati relativo all'esperimento selezionato, ne adatta il formato all'applet che svolgerà l'operazione e invia a quest'ultima il file.

3.2.12.4 Output

- Tutte le analisi consentite dall'applet sul file.

3.2.13 Logout.

3.2.13.1 Introduzione

Funzione che permette all'utente/amministratore di abbandonare l'applicazione e di rilasciare le risorse impegnate.

3.2.13.2 Input

L'utente seleziona l'opzione nel menu di navigazione dell'applicazione.

3.2.13.3 Funzione

- Rilascia le risorse impegnate dall'utente corrente.
- Registrazione dei dati in archivio che tengono traccia del logout;

3.2.13.4 Output

- Pagina di login

3.2.14 Visualizza dati personali.

3.2.14.1 Introduzione

Funzione che permette all'utente di richiedere la visualizzazione dei suoi dati personali di registrazione.

3.2.14.2 Input

L'utente seleziona l'opzione nel menu di navigazione dell'applicazione.

Dati

3.2.14.3 Funzione

- Visualizza i dati relativi all'utente che ha richiesto l'operazione.

3.2.14.4 Output

- Dati richiesti.

3.2.15 Modifica dati personali.

3.2.15.1 Introduzione

Funzione che permette all'utente di richiedere la visualizzazione dei suoi dati personali di registrazione per effettuare delle modifiche sui dati anagrafici.

3.2.15.2 Input

L'utente seleziona l'opzione nel menu di navigazione dell'applicazione e visualizzato l'elenco decide le modifiche da apportare.

Dati

- nome (obbligatorio) ,
- cognome(obbligatorio),
- indirizzo (obbligatorio),
- città (obbligatorio)
- Cap,
- data di nascita
- professione,
- indirizzo e-mail(obbligatorio),

- conferma indirizzo e-mail((obbligatorio),
- sesso(obbligatorio),

3.2.15.3 Funzione

- Visualizza i dati relativi all'utente che ha richiesto l'operazione.
- Controlla la presenza di valori nei campi obbligatori dopo le modifiche.
- Controlla la validità del formato nei campi modificati.

3.2.15.4 Output

Messaggi

"Modifiche effettuate con successo"

"Dati obbligatori non digitati"

3.2.16 Modifica dati accesso e dell'indirizzo email.

3.2.16.1 Introduzione

Funzione che permette all'utente/amministratore di richiedere la modifica dei suoi dati di "accesso" all'applicazione e dell'indirizzo email.

3.2.16.2 Input

L'utente/amministratore seleziona l'opzione nel menu di navigazione dell'applicazione e visualizzato l'elenco decide le modifiche da apportare.

Dati

- Username (obbligatorio) ,
- Password(obbligatorio),
- Gruppo di ricerca di appartenenza(obbligatorio),
- E-mail (obbligatorio),

3.2.16.3 Funzione

- Visualizza i dati di accesso relativi all'utente che ha richiesto l'operazione.

- Controlla la presenza di valori nei campi obbligatori dopo le modifiche.
- Controlla la validità del formato nei campi modificati.
- Pone l'utente in stato di attesa.

3.2.16.4 Output

Messaggi

"Modifiche inviate con successo,attendi una conferma da parte dell'amministratore".

"Dati obbligatori non digitati".

"L'e-mail non ha un formato valido".

3.2.17 Selezione della lingua di interesse per l'applicazione.

3.2.17.1 Introduzione

Funzione che permette all'utente/amministratore di selezionare la lingua di interesse nella quale navigare le pagine dell'applicazione.

3.2.17.2 Input

L'utente/amministratore seleziona l'opzione nel menu di navigazione dell'applicazione e visualizzato l'elenco decide la lingua da selezionare.

Dati

- Lingua selezionata (obbligatorio) ,

3.2.17.3 Funzione

- Visualizza le lingue disponibili per la visualizzazione delle pagine dell'applicazione.

3.2.17.4 Output

- Pagine dell'applicazione scritte nella lingua selezionata

2) Classe Amministratore

3.2.18 Aggiunta Gruppo di ricerca all'applicazione.

3.2.18.1 Introduzione

Permette all' amministratore di aggiungere un nuovo gruppo di ricerca a quelli disponibili in archivio.

3.2.18.2 Input

Form che permette l'inserimento del gruppo di ricerca nell'archivio.

Dati

- Gruppo di ricerca (obbligatorio).

3.2.18.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Controlla che il gruppo di ricerca non sia già presente nell'archivio.
- Registra il gruppo di ricerca in archivio.

3.2.18.4 Output

Form per l'aggiunta del gruppo di ricerca.

Messaggi

"Gruppo di ricerca non inserito".

"Gruppo di ricerca già presente nell'archivio".

"Gruppo di ricerca inserito con successo".

3.2.19 Eliminazione Gruppo di ricerca dall'applicazione.

3.2.19.1 Introduzione

Permette all' amministratore di eliminare un gruppo di ricerca da quelli disponibili in archivio.

3.2.19.2 Input

Form che permette la selezione del gruppo di ricerca da un menu a tendina.

Dati

- Gruppo di ricerca (obbligatorio).

3.2.19.3 Funzione

- Visualizza un Form per i dati in ingresso.

3.2.19.4 Output

Form per l'aggiunta del gruppo di ricerca.

Messaggi

"Eliminazione gruppo di ricerca avvenuta con successo".

3.2.20 Aggiunta nuova piattaforma all'applicazione.

3.2.20.1 Introduzione

Permette all' amministratore di aggiungere una nuova piattaforma a quelle disponibili in archivio.

3.2.20.2 Input

Form che permette l'inserimento della piattaforma nell'archivio.

Dati

- Tipo di piattaforma (obbligatorio).
- Nome della piattaforma (obbligatorio).
- Numero di righe
- Numero di colonne
- Numero di features

3.2.20.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Registra i dati della nuova piattaforma in archivio.

3.2.20.4 Output

Messaggi

"Dati obbligatori non inseriti".

"Piattaforma inserita con successo".

3.2.21 Aggiunta nuova sigla di piattaforma all'applicazione.

3.2.21.1 Introduzione

Permette all' amministratore di aggiungere la sigla di una nuova piattaforma a quelle disponibili in archivio.

3.2.21.2 Input

Form che permette l'inserimento della sigla della piattaforma nell'archivio.

Dati

- Nome della piattaforma (obbligatorio).

3.2.21.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Registra i dati della nuova piattaforma in archivio.

3.2.21.4 Output

- "Dati obbligatori non inseriti".
- "Nuova sigla inserita con successo".

3.2.22 Aggiunta nuovo tipo di piattaforma all'applicazione.

3.2.22.1 Introduzione

Permette all' amministratore di aggiungere un nuovo tipo di piattaforma a quelli disponibili in archivio.

3.2.22.2 Input

Form che permette l'inserimento del tipo della piattaforma nell'archivio.

Dati

- Tipo della piattaforma (obbligatorio).

3.2.22.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Registra i dati della nuova piattaforma in archivio.

3.2.22.4 Output

Messaggi

"Dati obbligatori non inseriti".

"Nuovo tipo di piattaforma inserito con successo".

3.2.23 Visualizza dati piattaforme.

3.2.23.1 Introduzione

Permette all' amministratore di visualizzare i dati delle piattaforme disponibili in archivio.

3.2.23.2 Input

L'utente seleziona l'opzione nel menu di navigazione dell'applicazione.

3.2.23.3 Funzione

- Visualizza i dati delle piattaforme registrate.

3.2.23.4 Output

Lista piattaforme e relativi dati.

Messaggi

"Nessuna piattaforma registrata".

3.2.24 Modifica dati piattaforma.

3.2.24.1 Introduzione

Permette all' amministratore di modificare i dati di una piattaforma.

3.2.24.2 Input

Il nome della piattaforma da modificare.

Dati

- Tipo di piattaforma (obbligatorio).
- Nome della piattaforma (obbligatorio).
- Numero di righe
- Numero di colonne
- Numero di features

3.2.24.3 Funzione

- Visualizza i dati delle piattaforme.
- Consente la modifica dei dati.
- Controlla la presenza dei dati obbligatori.
- Controlla la validità del formato dei dati inseriti.

3.2.24.4 Output

Dati Piattaforma

Messaggi

"Dati piattaforma modificati con successo".

"Dati obbligatori non inseriti".

3.2.25 Elimina dati piattaforma.

3.2.25.1 Introduzione

Permette all' amministratore di eliminare i dati di una piattaforma.

3.2.25.2 Input

Il nome della piattaforma da eliminare.

Dati

- Nome della piattaforma (obbligatorio).

3.2.25.3 Funzione

- Visualizza i dati della piattaforma.
- Consente l'eliminazione dei dati.

3.2.25.4 Output

Dati Piattaforma

Messaggi

"Eliminazione Piattaforma avvenuta con successo".

3.2.26 Cerca utente

3.2.26.1 Introduzione

Questa funzione permette ad un amministratore di cercare un utente registrato.

3.2.26.2 Input

Form per l'inserimento della Username.

Dati

- Username (obbligatorio).

3.2.26.3 Funzione

- Visualizza un Form per i parametri di ricerca in ingresso.
- Controlla che la username sia stata digitata, altrimenti visualizza un messaggio di errore.
  - Cerca in archivio utenti l'utente corrispondente alla username digitata.
  - Se è presente un utente con questa username, ne legge i dati dall'archivio.
  - Se non vi è nessun utente visualizza un messaggio di ricerca senza esito.

3.2.26.4 Output

Lista utenti con username inserita.

Messaggi

3.2.27 Interdizione utente

3.2.27.1 Introduzione

Questa funzione permette ad un amministratore di interdire un utente registrato.

3.2.27.2 Input

- Username e nome gruppo dell'utente da interdire.
- Username (obbligatorio).
- Gruppo di ricerca (obbligatorio).

3.2.27.3 Funzione

- Controlla che i campi obbligatori siano inseriti.
- Controlla la validità dei dati inseriti.
- L'utente prescelto è interdetto e qualsiasi suo accesso viene impedito.

3.2.27.4 Output

Utente selezionato.

Messaggi

"Utente interdetto".

3.2.28 Conferma utente

3.2.28.1 Introduzione

Questa funzione permette ad un amministratore di rendere attivo un utente registrato per la prima volta nell'applicazione.

3.2.28.2 Input

- Username e nome gruppo dell'utente da confermare.
- Username (obbligatorio).
- Gruppo di ricerca (obbligatorio).

3.2.28.3 Funzione

- Controlla che i campi obbligatori siano inseriti.
- Controlla la validità dei dati inseriti.
- L'utente prescelto è interdetto e qualsiasi suo accesso viene impedito.

3.2.28.4 Output

Utente selezionato.

Messaggi

"Utente confermato".

3.2.29 Sblocco utente

3.2.29.1 Introduzione

Questa funzione permette ad un amministratore di sbloccare un utente registrato precedentemente interdetto.

3.2.29.2 Input

- Username e nome gruppo dell'utente da sbloccare.
- Username (obbligatorio).
- Gruppo di ricerca (obbligatorio).

3.2.29.3 Funzione

- Controlla la validità dei dati inseriti.
- L'utente prescelto è sbloccato dall'interdizione.

3.2.29.4 Output

Utente selezionato.

Messaggi

"Utente sbloccato".

3.2.30 Promozione utente

3.2.30.1 Introduzione

Questa funzione permette ad un amministratore di promuovere un utente registrato ad amministratore del sistema .

3.2.30.2 Input

- Username e nome gruppo dell'utente da eliminare.
- Username (obbligatorio).
- Gruppo di ricerca (obbligatorio).

3.2.30.3 Funzione

- Controlla la validità dei dati inseriti.
- L'utente prescelto è promosso ad amministratore dall'applicazione.

3.2.30.4 Output

Utente selezionato.

Messaggi

"Utente promosso ad amministratore".

3.2.31 Eliminazione utente

3.2.31.1 Introduzione

Questa funzione permette ad un amministratore di eliminare un utente registrato.

3.2.31.2 Input

- Username e nome gruppo dell'utente da eliminare.
- Username(obbligatorio).

- Gruppo di ricerca(obbligatorio).

3.2.31.3 Funzione

- Controlla che i campi obbligatori siano inseriti.
- Controlla la validità dei dati inseriti.
- L'utente prescelto è eliminato dall'applicazione.

3.2.31.4 Output

Utente selezionato.

Messaggi

"Utente eliminato".

3.2.32 Aggiunta Test statistico.

3.2.32.1 Introduzione

Permette all' amministratore di aggiungere un nuovo test statistico a quelli disponibili in archivio.

3.2.32.2 Input

Form che permette l'inserimento del test statistico nell'archivio.

Dati

- Test statistico (obbligatorio).

3.2.32.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Controlla che il test statistico non sia già presente nell'archivio.
- Registra il test statistico in archivio.

3.2.32.4 Output

- Form per l'aggiunta del test statistico.

  Messaggi

  "Test statistico non inserito".

  "Test statistico già presente nell'archivio".

  "Test statistico inserito con successo".

3.2.33 Eliminazione Test statistico.

3.2.33.1 Introduzione

Permette all' amministratore di eliminare un test statistico da quelli disponibili in archivio.

3.2.33.2 Input

Form che permette l'eliminazione del test statistico dall'archivio tramite un menu a tendina..

Dati

- Test statistico (obbligatorio).

3.2.33.3 Funzione

- Visualizza un Form per i dati in ingresso.

3.2.33.4 Output

Form per l'aggiunta del test statistico.

Messaggi

"Eliminazione Test statistico avvenuta con successo".

3.2.34 Aggiunta Parametro statistico.

3.2.34.1 Introduzione

Permette all' amministratore di aggiungere un nuovo parametro statistico a quelli disponibili in archivio.

3.2.34.2 Input

Form che permette l'inserimento del parametro statistico nell'archivio.

Dati

- Parametro statistico (obbligatorio).

3.2.34.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Controlla che il parametro statistico non sia già presente nell'archivio.

- Registra il parametro statistico in archivio.

3.2.34.4 Output

Form per l'aggiunta del parametro statistico.

Messaggi

"Parametro statistico non inserito".

"Parametro statistico già presente nell'archivio".

"Parametro statistico inserito con successo".

3.2.35 Eliminazione Parametro statistico.

3.2.35.1 Introduzione

Permette all' amministratore di eliminare un parametro statistico da quelli disponibili in archivio.

3.2.35.2 Input

Form che permette l'eliminazione del parametro statistico dall'archivio tramite un menu a tendina..

Dati

- Parametro statistico (obbligatorio).

3.2.35.3 Funzione

- Visualizza un Form per i dati in ingresso.

3.2.35.4 Output

Form per l'eliminazione del parametro statistico.

Messaggi

"Eliminazione parametro statistico avvenuta con successo".

3.2.36 Aggiunta tipologia esperimento.

3.2.36.1 Introduzione

Permette all' amministratore di aggiungere una nuova tipologia di esperimento a quelle disponibili in archivio.

3.2.36.2 Input

Form che permette l'inserimento della tipologia di esperimento nell'archivio.

Dati

- Tipologia di esperimento (obbligatorio).

3.2.36.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Controlla che la tipologia di esperimento non sia già presente nell'archivio.
- Registra la tipologia di esperimento in archivio.

3.2.36.4 Output

Form per l'aggiunta della tipologia di esperimento.

Messaggi

"Tipologia di esperimento non inserita".

"Tipologia di esperimento già presente nell'archivio".

"Tipologia di esperimento inserita con successo".

3.2.37 Eliminazione tipologia esperimento.

3.2.37.1 Introduzione

Permette all' amministratore di eliminare una tipologia di esperimento da quelle disponibili in archivio.

3.2.37.2 Input

Form che permette l'eliminazione della tipologia di esperimento dall'archivio da un menu a tendina.

Dati

- Tipologia di esperimento (obbligatorio).

3.2.37.3 Funzione

Visualizza un Form per i dati in ingresso.

3.2.37.4 Output

Form per l'eliminazione della tipologia di esperimento.

Messaggi

"Cancellazione tipologia di esperimento avvenuta con successo".

3.2.38 Aggiunta laboratorio.

3.2.38.1 Introduzione

Permette all' amministratore di aggiungere un nuovo laboratorio a quelli disponibili in archivio.

3.2.38.2 Input

Form che permette l'inserimento del laboratorio nell'archivio.

Dati

- Laboratorio (obbligatorio).

3.2.38.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Controlla che il laboratorio non sia già presente nell'archivio.
- Registra il laboratorio in archivio.

3.2.38.4 Output

Form per l'aggiunta del laboratorio in archivio.

Messaggi

"Laboratorio non inserito".

"Laboratorio già presente nell'archivio".

"Laboratorio inserito con successo".

3.2.39 Eliminazione laboratorio.

3.2.39.1 Introduzione

Permette all' amministratore di eliminare un laboratorio da quelli disponibili in archivio.

3.2.39.2 Input

Form che permette l'eliminazione del laboratorio nell'archivio tramite un menu a tendina.

Dati

- Laboratorio (obbligatorio).

3.2.39.3 Funzione

Visualizza un Form per i dati in ingresso.

3.2.39.4 Output

Form per l'eliminazione del laboratorio dall'archivio.

Messaggi

"Eliminazione laboratorio avvenuta con successo".

3.2.40 Aggiunta organismo.

3.2.40.1 Introduzione

Permette all' amministratore di aggiungere un nuovo organismo a quelli disponibili in archivio.

3.2.40.2 Input

Form che permette l'inserimento dell'organismo nell'archivio.

Dati

- Organismo (obbligatorio).

3.2.40.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Controlla che l'organismo non sia già presente nell'archivio.
- Registra l'organismo in archivio.

3.2.40.4 Output

Form per l'aggiunta dell'organismo in archivio.

Messaggi

"Organismo non inserito".

"Organismo già presente nell'archivio".

"Organismo inserito con successo".

3.2.41 Eliminazione organismo.

3.2.41.1 Introduzione

Permette all' amministratore di eliminare un organismo da quelli disponibili in archivio.

3.2.41.2 Input

Form che permette l'eliminazione dell'organismo nell'archivio tramite un menu a tendina.

Dati

- Organismo (obbligatorio).

3.2.41.3 Funzione

Visualizza un Form per i dati in ingresso.

3.2.41.4 Output

Form per l'eliminazione dell'organismo dall'archivio.

Messaggi

"Eliminazione organismo avvenuta con successo".

3.2.42 Aggiunta fattore di correzione.

3.2.42.1 Introduzione

Permette all' amministratore di aggiungere un nuovo fattore di correzione a quelli disponibili in archivio.

3.2.42.2 Input

Form che permette l'inserimento del fattore di correzione nell'archivio.

Dati

- Fattore di correzione (obbligatorio).

3.2.42.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.

- Controlla che il fattore di correzione non sia già presente nell'archivio.
- Registra il fattore di correzione in archivio.

3.2.42.4 Output

Form per l'aggiunta del fattore di correzione in archivio.

Messaggi

"Fattore di correzione non inserito".

"Fattore di correzione già presente nell'archivio".

"Fattore di correzione inserito con successo".

3.2.43 Eliminazione fattore di correzione.

3.2.43.1 Introduzione

Permette all' amministratore di eliminare un fattore di correzione da quelli disponibili in archivio.

3.2.43.2 Input

Form che permette l'eliminazione del fattore di correzione dall'archivio tramite un menu a tendina.

Dati

- Fattore di correzione (obbligatorio).

3.2.43.3 Funzione

Visualizza un Form per i dati in ingresso.

3.2.43.4 Output

Form per l'eliminazione del fattore di correzione dall'archivio.

Messaggi

"Eliminazione fattore di correzione avvenuta con successo".

3.2.44 Aggiunta fattore sperimentale.

3.2.44.1 Introduzione

Permette all' amministratore di aggiungere un nuovo fattore sperimentale a quelli disponibili in archivio.

3.2.44.2 Input

Form che permette l'inserimento del fattore sperimentale nell'archivio.

Dati

- Fattore sperimentale (obbligatorio).

3.2.44.3 Funzione

- Visualizza un Form per i dati in ingresso.
- Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
- Controlla che il fattore sperimentale non sia già presente nell'archivio.
- Registra il fattore sperimentale in archivio.

3.2.44.4 Output

Form per l'aggiunta del fattore sperimentale in archivio.

Messaggi

"Fattore sperimentale non inserito".

"Fattore sperimentale già presente nell'archivio".

"Fattore sperimentale inserito con successo".

3.2.45 Eliminazione fattore sperimentale.

3.2.45.1 Introduzione

Permette all' amministratore di eliminare un fattore di correzione da quelli disponibili in archivio.

3.2.45.2 Input

Form che permette l'eliminazione del fattore di correzione dall'archivio tramite un menu a tendina.

Dati

- Fattore di correzione (obbligatorio).

3.2.45.3 Funzione

Visualizza un Form per i dati in ingresso.

3.2.45.4 Output

Form per l'eliminazione del fattore di correzione dall'archivio.

Messaggi

"Eliminazione fattore di correzione avvenuta con successo".

3) Classe Visitatore

3.2.46 Richiesta registrazione

3.2.46.1 Introduzione

Permette ad un visitatore di registrarsi per accedere ai servizi di "MicroArrayWeb" offerti ad un utente registrato.

3.2.46.2 Input

Dati personali:

- nome (obbligatorio) ,
- cognome (obbligatorio),
- indirizzo (obbligatorio),
- città (obbligatorio),
- cap (obbligatorio),
- data di nascita (obbligatorio),
- professione,
- indirizzo e-mail (obbligatorio),
- username (obbligatorio),
- password (obbligatorio),
- conferma password (obbligatorio),
- gruppo di ricerca di appartenenza(obbligatorio),
- sesso(obbligatorio),

3.2.46.3 Funzione

Visualizzazione di un form per l'input dei dati, sui quali il sistema esegue le seguenti operazioni :

- Controlla che i campi relativi all'Email, Username, Password,Conferma Password, Nome, Cognome, Indirizzo, Città,Cap non siano vuoti o non validi, in particolare l'Email abbia una struttura corretta e username e password abbiano una lunghezza di almeno 5 caratteri;
- Controlla che la coppia Username-Gruppo di ricerca non sia già presente in archivio;
- Controlla che la Password e la Conferma Password siano uguali;
- Verifica che l'indirizzo e-mail non appartenga ad un utente già registrato.
- Registrazione dei dati di input in archivio;

3.2.46.4 Output

Messaggi

"Confermi la registrazione?"

"Registrazione effettuata con successo".

"Email non valida nel formato".

"Il campo relativo all'email non può essere vuoto".

"Il campo relativo alla Username non può essere vuoto".

"Il campo relativo alla Password non può essere vuoto".

"La Password deve essere almeno di 5 caratteri".

"Le 2 password inserite sono diverse".

"Dati obbligatori non digitati".

"Indirizzo e-mail già in uso da un altro utente".

"Esiste già un utente con questa username e questa password."

Tutti i dati letti in input registrati in archivio.

Menù di accesso alle aree di utente.

Altri dati registrati :

- Data/Ora registrazione

3.3 Requisiti di prestazioni

Nessuno.

3.4 Vincoli di progetto

Nessuno.

3.5 Attributi

3.5.1 Sicurezza

Poichè le informazioni elaborate da questo sistema software sono di importanza rilevante per la struttura in cui sarà adoperato,non è sufficiente come strumento di protezione esclusivamente l'utilizzo di una password al fine di evitare accessi al sistema da parte di persone esterne all’ambiente.

E’ stato infatti necessario aggiungere il concetto di ambito di visibilità di un esperimento, per poter garantire l’accesso dell’utente ai soli esperimenti per i quali è permessa la sua consultazione.

3.6 Altri requisiti

3.6.1 Database

Il sistema software prevede l’utilizzo di un database relazionale per l’archiviazione dei dati.

APPENDICE B

Creazione ed inizializzazione della base di dati (Script generato dal CASE Power Designer 11)

```
-- MySQL Administrator dump 1.4
--
-- ------------------------------------------------------
-- Server version       5.0.18-nt


/*!40101 SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
--
-- Create schema microarray
--
CREATE DATABASE IF NOT EXISTS microarray;
USE microarray;
--
```

```
-- Definition of table `analisi`
--
DROP TABLE IF EXISTS `analisi`;
CREATE TABLE `analisi` (
  `IDANALISI` int(11) NOT NULL auto_increment,
  `ID_` int(11) NOT NULL,
  `NOMETEST` varchar(30) NOT NULL,
  `CAMPIONEPROVA` varchar(100) default NULL,
  `CAMPIONERIFERIMENTO` varchar(100) default NULL,
  `LISTAGENIREGOLATI` longblob,
  PRIMARY KEY  (`IDANALISI`),
  KEY `RELATIONSHIP_2_FK` (`NOMETEST`),
  KEY `RELATIONSHIP_4_FK` (`ID_`),
  CONSTRAINT `FK_RELATIONSHIP_2` FOREIGN KEY (`NOMETEST`)
REFERENCES `teststatistico` (`NOMETEST`) ON DELETE CASCADE ON
UPDATE CASCADE,
  CONSTRAINT `FK_RELATIONSHIP_4` FOREIGN KEY (`ID_`) REFERENCES
`esperimento` (`ID_`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
--
-- Dumping data for table `analisi`
--
/*!40000 ALTER TABLE `analisi` DISABLE KEYS */;
/*!40000 ALTER TABLE `analisi` ENABLE KEYS */;


--
-- Definition of table `analisistatistica`
--
```

```
DROP TABLE IF EXISTS `analisistatistica`;
CREATE TABLE `analisistatistica` (
  `IDPARAMETRO` int(11) NOT NULL auto_increment,
  `IDANALISI` int(11) NOT NULL,
  `PARAMETRO` varchar(40) NOT NULL,
  `VALORE` varchar(30) default NULL,
  PRIMARY KEY  (`IDPARAMETRO`),
  KEY `RELATIONSHIP_3_FK` (`IDANALISI`),
  KEY `RELATIONSHIP_5_FK` (`PARAMETRO`),
  CONSTRAINT `FK_RELATIONSHIP_3` FOREIGN KEY (`IDANALISI`)
REFERENCES `analisi` (`IDANALISI`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `FK_RELATIONSHIP_5` FOREIGN KEY (`PARAMETRO`)
REFERENCES `parametristatistici` (`PARAMETRO`) ON DELETE CASCADE ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `analisistatistica`
--

/*!40000 ALTER TABLE `analisistatistica` DISABLE KEYS */;
/*!40000 ALTER TABLE `analisistatistica` ENABLE KEYS */;

--
-- Definition of table `array`
--

DROP TABLE IF EXISTS `array`;
CREATE TABLE `array` (
  `ID_ARRAY` int(11) NOT NULL auto_increment,
  `SIGLA` varchar(100) NOT NULL,
```

```
  `TIPOPIATTAFORMA` varchar(40) NOT NULL,
  `RIGHE` int(11) default NULL,
  `COLONNE` int(11) default NULL,
  `NUMEROFEATURES` int(11) default NULL,
  PRIMARY KEY  (`ID_ARRAY`),
  KEY `RELATIONSHIP_11_FK` (`SIGLA`),
  KEY `RELATIONSHIP_12_FK` (`TIPOPIATTAFORMA`),
  CONSTRAINT `FK_RELATIONSHIP_11` FOREIGN KEY (`SIGLA`)
REFERENCES `nomearray` (`SIGLA`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `FK_RELATIONSHIP_12` FOREIGN KEY
(`TIPOPIATTAFORMA`) REFERENCES `tipologiapiattaforma`
(`TIPOPIATTAFORMA`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--
-- Dumping data for table `array`
--
/*!40000 ALTER TABLE `array` DISABLE KEYS */;
/*!40000 ALTER TABLE `array` ENABLE KEYS */;
--
-- Definition of table `campione`
--
DROP TABLE IF EXISTS `campione`;
CREATE TABLE `campione` (
  `ID_CAMPIONE` int(11) NOT NULL auto_increment,
  `ID_` int(11) NOT NULL,
  `NOMEORGANISMO` varchar(30) NOT NULL,
  `NOME` varchar(50) default NULL,
  `PROTOCOLLOTRATTAMENTO` text,
  `METODOESTRAZIONE` text,
  `TIPOAMPLIFICAZIONE` text,
```

```
  `PROTOCOLLOIBRIDAZIONE` text,
  `PROTOCOLLOMARCATURA` text,
  PRIMARY KEY  (`ID_CAMPIONE`),
  KEY `RELATIONSHIP_6_FK` (`ID_`),
  KEY `RELATIONSHIP_8_FK` (`NOMEORGANISMO`),
  CONSTRAINT `FK_RELATIONSHIP_6` FOREIGN KEY (`ID_`) REFERENCES
`esperimento` (`ID_`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_RELATIONSHIP_8` FOREIGN KEY (`NOMEORGANISMO`)
REFERENCES `organismo` (`NOMEORGANISMO`) ON DELETE CASCADE ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--
-- Dumping data for table `campione`
--
/*!40000 ALTER TABLE `campione` DISABLE KEYS */;
/*!40000 ALTER TABLE `campione` ENABLE KEYS */;
--
-- Definition of table `esperimento`
--
DROP TABLE IF EXISTS `esperimento`;
CREATE TABLE `esperimento` (
  `ID_` int(11) NOT NULL auto_increment,
  `FATTORE` varchar(40) NOT NULL,
  `SIGLA` varchar(100) NOT NULL,
  `NOMELABORATORIO` varchar(40) NOT NULL,
  `NOMEGRUPPO` varchar(30) NOT NULL,
  `USERNAME` varchar(30) NOT NULL,
  `TIPOESPERIMENTO` varchar(50) NOT NULL,
  `BREVEDESCRIZIONE` text NOT NULL,
  `NUMEROIBRIDAZIONI` smallint(6) default NULL,
  `AUTORE` varchar(40) NOT NULL,
```

```
  `PROTOCOLLOSCANSIONE` text,
  `SOFTWAREANALISI` text,
  `TIPONORMALIZZAZIONE` text,
  `SCALALOGARITMICA` tinyint(1) default NULL,
  `NUMEROFATTORISPERIMENTALI` smallint(6) default NULL,
  `FATTORISPERIMENTALI` text,
  `VISIBILITA` varchar(20) NOT NULL,
  `CONTATTI` text,
  `LINKS_URL` text,
  `CONTROLLIQUALITA` text,
  `TABELLAANNOTAZIONI` longblob,
  `DATIGREZZI` longblob,
  `DATINORMALIZZATI` longblob,
  `DATAINSERIMENTO` timestamp NOT NULL default CURRENT_TIMESTAMP
on update CURRENT_TIMESTAMP,
  PRIMARY KEY  (`ID_`),
  KEY `RELATIONSHIP_10_FK` (`SIGLA`),
  KEY `RELATIONSHIP_13_FK` (`TIPOESPERIMENTO`),
  KEY `RELATIONSHIP_14_FK` (`NOMELABORATORIO`),
  KEY `RELATIONSHIP_15_FK` (`FATTORE`),
  KEY `RELATIONSHIP_16_FK` (`NOMEGRUPPO`,`USERNAME`),
  CONSTRAINT `FK_RELATIONSHIP_10` FOREIGN KEY (`SIGLA`)
REFERENCES `nomearray` (`SIGLA`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `FK_RELATIONSHIP_13` FOREIGN KEY
(`TIPOESPERIMENTO`) REFERENCES `tipologiaesperimento`
(`TIPOESPERIMENTO`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_RELATIONSHIP_14` FOREIGN KEY
(`NOMELABORATORIO`) REFERENCES `laboratorio` (`NOMELABORATORIO`)
ON DELETE CASCADE ON UPDATE CASCADE,
```

```
  CONSTRAINT `FK_RELATIONSHIP_15` FOREIGN KEY (`FATTORE`)
REFERENCES `fattorecorrezione` (`FATTORE`) ON DELETE CASCADE ON
UPDATE CASCADE,
  CONSTRAINT `FK_RELATIONSHIP_16` FOREIGN KEY (`NOMEGRUPPO`,
`USERNAME`) REFERENCES `utente` (`NOMEGRUPPO`, `USERNAME`) ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `esperimento`
--
/*!40000 ALTER TABLE `esperimento` DISABLE KEYS */;
/*!40000 ALTER TABLE `esperimento` ENABLE KEYS */;


--
-- Definition of table `fattorecorrezione`
--

DROP TABLE IF EXISTS `fattorecorrezione`;
CREATE TABLE `fattorecorrezione` (
  `FATTORE` varchar(40) NOT NULL,
  PRIMARY KEY  (`FATTORE`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `fattorecorrezione`
--

/*!40000 ALTER TABLE `fattorecorrezione` DISABLE KEYS */;
INSERT INTO `fattorecorrezione` (`FATTORE`) VALUES
 ('BenjaminHochberg'),
```

```
 ('Bonferroni');
/*!40000 ALTER TABLE `fattorecorrezione` ENABLE KEYS */;

--
-- Definition of table `fattoresperimentale`
--

DROP TABLE IF EXISTS `fattoresperimentale`;
CREATE TABLE `fattoresperimentale` (
  `IDFATTORE` int(11) NOT NULL auto_increment,
  `FATTORESPERIMENTALE` varchar(50) NOT NULL,
  `ID_CAMPIONE` int(11) NOT NULL,
  `VALORE` varchar(30) default NULL,
  PRIMARY KEY  (`IDFATTORE`),
  KEY `RELATIONSHIP_7_FK` (`ID_CAMPIONE`),
  KEY `RELATIONSHIP_9_FK` (`FATTORESPERIMENTALE`),
  CONSTRAINT `FK_RELATIONSHIP_7` FOREIGN KEY (`ID_CAMPIONE`)
REFERENCES `campione` (`ID_CAMPIONE`) ON DELETE CASCADE ON
UPDATE CASCADE,
  CONSTRAINT `FK_RELATIONSHIP_9` FOREIGN KEY
(`FATTORESPERIMENTALE`) REFERENCES `listafattorisperimentali`
(`FATTORESPERIMENTALE`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `fattoresperimentale`
--

/*!40000 ALTER TABLE `fattoresperimentale` DISABLE KEYS */;
/*!40000 ALTER TABLE `fattoresperimentale` ENABLE KEYS */;
```

```
--
-- Definition of table `grupporicerca`
--

DROP TABLE IF EXISTS `grupporicerca`;
CREATE TABLE `grupporicerca` (
  `NOMEGRUPPO` varchar(30) NOT NULL,
  PRIMARY KEY  (`NOMEGRUPPO`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `grupporicerca`
--

/*!40000 ALTER TABLE `grupporicerca` DISABLE KEYS */;
INSERT INTO `grupporicerca` (`NOMEGRUPPO`) VALUES
 ('Studente');
/*!40000 ALTER TABLE `grupporicerca` ENABLE KEYS */;



--
-- Definition of table `laboratorio`
--

DROP TABLE IF EXISTS `laboratorio`;
CREATE TABLE `laboratorio` (
  `NOMELABORATORIO` varchar(40) NOT NULL,
  PRIMARY KEY  (`NOMELABORATORIO`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
```

```
-- Dumping data for table `laboratorio`
--

/*!40000 ALTER TABLE `laboratorio` DISABLE KEYS */;
INSERT INTO `laboratorio` (`NOMELABORATORIO`) VALUES
 ('cnr');
/*!40000 ALTER TABLE `laboratorio` ENABLE KEYS */;


--
-- Definition of table `listafattorisperimentali`
--

DROP TABLE IF EXISTS `listafattorisperimentali`;
CREATE TABLE `listafattorisperimentali` (
  `FATTORESPERIMENTALE` varchar(50) NOT NULL,
  PRIMARY KEY  (`FATTORESPERIMENTALE`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `listafattorisperimentali`
--

/*!40000 ALTER TABLE `listafattorisperimentali` DISABLE KEYS */;
INSERT INTO `listafattorisperimentali` (`FATTORESPERIMENTALE`) VALUES
 ('cell line'),
 ('dose'),
 ('genetic variation'),
 ('time'),
 ('tissue'),
 ('treatment');
/*!40000 ALTER TABLE `listafattorisperimentali` ENABLE KEYS */;
```

```
--
-- Definition of table `nomearray`
--

DROP TABLE IF EXISTS `nomearray`;
CREATE TABLE `nomearray` (
  `SIGLA` varchar(100) NOT NULL,
  PRIMARY KEY  (`SIGLA`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `nomearray`
--

/*!40000 ALTER TABLE `nomearray` DISABLE KEYS */;
INSERT INTO `nomearray` (`SIGLA`) VALUES
 ('Affymetrix111');
/*!40000 ALTER TABLE `nomearray` ENABLE KEYS */;

--
-- Definition of table `organismo`
--

DROP TABLE IF EXISTS `organismo`;
CREATE TABLE `organismo` (
  `NOMEORGANISMO` varchar(30) NOT NULL,
  PRIMARY KEY  (`NOMEORGANISMO`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
```

```
-- Dumping data for table `organismo`
--

/*!40000 ALTER TABLE `organismo` DISABLE KEYS */;
INSERT INTO `organismo` (`NOMEORGANISMO`) VALUES
 ('homo sapiens'),
 ('mus musculus'),
 ('rattus norvegieus');
/*!40000 ALTER TABLE `organismo` ENABLE KEYS */;

--
-- Definition of table `parametristatistici`
--

DROP TABLE IF EXISTS `parametristatistici`;
CREATE TABLE `parametristatistici` (
  `PARAMETRO` varchar(40) NOT NULL,
  PRIMARY KEY  (`PARAMETRO`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `parametristatistici`
--

/*!40000 ALTER TABLE `parametristatistici` DISABLE KEYS */;
INSERT INTO `parametristatistici` (`PARAMETRO`) VALUES
 ('delta'),
 ('foldchange'),
 ('p-value');
/*!40000 ALTER TABLE `parametristatistici` ENABLE KEYS */;
```

```
--
-- Definition of table `teststatistico`
--
DROP TABLE IF EXISTS `teststatistico`;
CREATE TABLE `teststatistico` (
  `NOMETEST` varchar(30) NOT NULL,
  PRIMARY KEY  (`NOMETEST`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `teststatistico`
--

/*!40000 ALTER TABLE `teststatistico` DISABLE KEYS */;
INSERT INTO `teststatistico` (`NOMETEST`) VALUES
 ('Illumina-Diffscore'),
 ('SAM'),
 ('t-Test');
/*!40000 ALTER TABLE `teststatistico` ENABLE KEYS */;

--
-- Definition of table `tipologiaesperimento`
--

DROP TABLE IF EXISTS `tipologiaesperimento`;
CREATE TABLE `tipologiaesperimento` (
  `TIPOESPERIMENTO` varchar(50) NOT NULL,
  PRIMARY KEY  (`TIPOESPERIMENTO`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `tipologiaesperimento`
```

```
--

/*!40000 ALTER TABLE `tipologiaesperimento` DISABLE KEYS */;
INSERT INTO `tipologiaesperimento` (`TIPOESPERIMENTO`) VALUES
 ('dose response'),
 ('effect of gene knock-out'),
 ('normal vs.diseased comparison'),
 ('time course'),
 ('treated vs.untreated comparison');
/*!40000 ALTER TABLE `tipologiaesperimento` ENABLE KEYS */;


--
-- Definition of table `tipologiapiattaforma`
--

DROP TABLE IF EXISTS `tipologiapiattaforma`;
CREATE TABLE `tipologiapiattaforma` (
  `TIPOPIATTAFORMA` varchar(40) NOT NULL,
  PRIMARY KEY (`TIPOPIATTAFORMA`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `tipologiapiattaforma`
--

/*!40000 ALTER TABLE `tipologiapiattaforma` DISABLE KEYS */;
INSERT INTO `tipologiapiattaforma` (`TIPOPIATTAFORMA`) VALUES
 ('Affymetrix'),
 ('Agilent'),
 ('Amershaw'),
 ('Illumina');
```

```
/*!40000 ALTER TABLE `tipologiapiattaforma` ENABLE KEYS */;

--
-- Definition of table `utente`
--

DROP TABLE IF EXISTS `utente`;
CREATE TABLE `utente` (
  `NOMEGRUPPO` varchar(30) NOT NULL,
  `USERNAME` varchar(30) NOT NULL,
  `NOME` varchar(50) NOT NULL,
  `COGNOME` varchar(50) NOT NULL,
  `DATANASCITA` date NOT NULL,
  `CITTA` varchar(50) NOT NULL,
  `INDIRIZZO` varchar(100) NOT NULL,
  `CAP` varchar(10) default NULL,
  `PROFESSIONE` varchar(40) default NULL,
  `EMAIL` varchar(40) default NULL,
  `PASSWORD` varchar(30) default NULL,
  `SESSO` varchar(5) default NULL,
  `TIPO` varchar(15) default 'utente',
  `LOGGATO` tinyint(1) default '0',
  `INTERDETTO` tinyint(1) default '0',
  `ATTESA` tinyint(1) default '1',
  `TENTATIVIRIMASTI` smallint(6) default '3',
  `DATAREGISTRAZIONE` timestamp NOT NULL default CURRENT_TIMESTAMP
on update CURRENT_TIMESTAMP,
  PRIMARY KEY  (`NOMEGRUPPO`,`USERNAME`),
  KEY `RELATIONSHIP_1_FK` (`NOMEGRUPPO`),
  CONSTRAINT `FK_RELATIONSHIP_1` FOREIGN KEY (`NOMEGRUPPO`)
REFERENCES `grupporicerca` (`NOMEGRUPPO`) ON DELETE CASCADE ON
UPDATE CASCADE
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
--
-- Dumping data for table `utente`
--

/*!40000 ALTER TABLE `utente` DISABLE KEYS */;
INSERT INTO `utente`
(`NOMEGRUPPO`,`USERNAME`,`NOME`,`COGNOME`,`DATANASCITA`,`CITTA
`,`INDIRIZZO`,`CAP`,`PROFESSIONE`,`EMAIL`,`PASSWORD`,`SESSO`,`TIPO`,`L
OGGATO`,`INTERDETTO`,`ATTESA`,`TENTATIVIRIMASTI`,`DATAREGISTRAZI
ONE`) VALUES
 ('studente','administrator','Giuseppe','Battinelli','1980-06-13','avellino','viale
umberto
nobile','83100','studente','gbattine@alice.it','administrator','uomo','amministratore',0
,0,0,3,'2006-12-01 11:24:35');
/*!40000 ALTER TABLE `utente` ENABLE KEYS */;


/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;
/*!40101 SET
CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION
*/;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;
```

Riferimenti Bibliografici e Webografici

Capitolo I-Introduzione

Ford N. - "*Art of Java Web Development*" - Ed. Manning
Mann K.D. - "*JavaServer Faces in Action*" - Ed. Manning
Pianciamore M. - "*Principi di Ingegneria del Software – Design Pattern*" - CEFRIEL
Mecca G.- "*Applicazioni Web J2EE : Java Servlet*", corso di Tecnologie di sviluppo per il Web - Università della Basilicata
Falkner J,Jones.K - "*Servlets and JavaServer Pages : The J2EE technology Web tier*" - Ed. Addison Wesley

Capitolo II-Microarray e fondamenti di genetica

Wolfe Stephen L., "*Biologia cellulare e molecolare*", Edises
Russell Peter J.,"*Genetica*", Edises
Brooker r. J., "*Genetica. Analisi e Principi*", *Zanichelli, 2000.*
Luscombe n. M., Greenbaum d., Gerstein m., "*What is Bioinformatics? A proposed definition and overview of the field*".
Farabee M. J., "*On Line biology book*", *2001, (http://www.emc.maricopa.edu/faculty/farabee/BIOBK/BioBookTOC.html)*
Brazma A., Parkinson H., Schlitt T., Shojatalab M., "*A quick introduction to elements of biology – cells, molecules, genes, functional genomics, microarrays*", EMBL-EBI, 2001, (http://www.ebi.ac.uk/microarray/biology_intro.html).
Mutarelli M., Facchiano A., Facchiano F., Colonna G."*Bioinformatics tools for the analysis of gene expression by microarrays*"

EBI "*Minimum information about a microarray experiment - MIAME*" (http://www.mged.org/Workgroups/MIAME/miame.html)

Capitolo III- Il framework Java Server Faces

Doench G, "*Core JavaServer Faces – 2004*" **-** Prentice Hall
Armstrong E., **Ball J.**, **Bodoff S.**, **Carson D.B.**, **Evans I.**, **Green D.**, **Haase K.**, **Jendrock E.** - "*The J2EE 1.4 Tutorial*" (cap. 17,18,19,20,21) - Sun Microsystems
Mann K.D. - "*JavaServer Faces in Action*" - Ed. Manning

Chen D. - "*JavaServer Faces and Sun Java Studio Creator :Experience the Next Generation Web-Application Framework*", Sun Tech Days 2005 - Sun Microsystems

Goyal D., **Varma V.**- "*Introduction to JavaServer Faces (JSF)*" – Sun Microsystems

Shin S. - "*J2EE Advanced*" (www.javapassion.com) - Sun Microsystems

Hall M. - Tutorial "*JavaServer Faces 1.1*" (www.coreservlets.com)

Pardi J."*JavaServer Faces*", 25 Maggio 2005

Rigamonti G. "JavaServer Faces Framework" (http://www.javaportal.it)

Hightower R. **"JSF for nonbelievers: The JSF application lifecycle" (http://www-128.ibm.com/developerworks/library/j-jsf2)**

"Framework, librerie, design Pattern" (http://www.make-it.it/java/librerie-java.php)

Capitolo IV-Progettazione dell'applicazione

Pressman R.S. - "*Principi di Ingegneria del Software*" - Ed.McGraw/Hill

Fowler M. – "*UML Distilled*" – Addison Wesley

Leszek A. – "*Sviluppo di sistemi informativi con UML*" - Addison Wesley

P. Atzeni, **S. Ceri**, **S. Paraboschi**, **R. Torlone** - "*Basi di dati*", seconda edizione - Ed. McGraw/Hill

Sybase - "*Sybase Power Designer : Conceptual Data Model , User's Guide*"

Sybase - "*Sybase Power Designer : Object Oriented Model , User's Guide*"

Sybase - "*Sybase Power Designer : Physical Data Model , User's Guide*"

Capitolo V-Implementazione dell'applicazione Microarray Web

B. Dudney - "*Creating JSF Custom Components*" (www.java.net)

R. Hightower - "*JSF for nonbelievers : JSF component development*" (www.ibm.com)

Apache Software Foundation - "*Apache Jakarta Tomcat 5 official documentation : JNDI Resources How-to*"

Apache Software Foundation - "*Apache Jakarta Tomcat 5 official documentation : JDBC Datasource How-to*"

McCallum E. *"JDBC Datasource con Commons DBCP"* (www.javastaff.com)

Astrologer J. *"Validating an Email Address in JSF"* (http://blogs.sun.com/johnc/entry/validating_an_email_address_in)

Troesser D. *"Apache Myfaces"* (http://www.ociweb.com/jnb/jnbJul2005.html)
Sun Microsystem"JavaMail Api Specification" (http://java.sun.com/products/javamail/reference/api/index.html)
Wolski S. "First Java Server Faces Tutoriale" (http://www.laliluna.de/first-java-server-faces-tutorial.html)
Poggiali G. "Gestire le Sessioni con le Servlet" (http://www.mokabyte.it/1999/01/httpsession.htm)
W3C Recommendation "Html 4.01 Specification (http://www.w3.org/TR/REC-html40)
Sun Microsystem *"J2SE 5.0 Demonstration Applets"* (http://java.sun.com/products/plugin/1.5.0/demos/plugin/applets.html)
Tabor P. "JSF for nonbelievers: JSF conversion and validation"(http://www-128.ibm.com/developerworks/java/library/j-jsf3/)
Vohra D. "Generating a JSF Data Table" (http://www.oracle.com/technology/oramag/oracle/06-jan/o16jsf.html)
Hall M. "JSF Tutorial" (http://www.coreservlets.com/JSF-Tutorial)
Larotonda A."La tecnologia JSP e gli scriptlet Java" (http://www.mokabyte.it/2004/12/jstl.htm)
Sun Microsystem *"Using the File Upload Component"* (http://developers.sun.com/prodtech/javatools/jscreator/learning/tutorials/2/file_upload.html)
Cioroianu A."Upload Files with JSF and MyFaces" (http://www.onjava.com)

BalusC **"Using datatables" (http://balusc.xs4all.nl/srv/dev-jep-dat.html)**

Zinovyev A. "Vimida Tutorial" (http://bioinfo-out.curie.fr/projects/vimida/tutorial.htm)
Jakarta Project *"Commons Email"* (http://jakarta.apache.org/commons/email)

Appendice A- SRS Microarray Web Application

Software Engineering Standards Committee of the IEEE Computer Society – *"IEEE Recommended Practice for Software Requirements Specifications"*, IEEE Std 830-1998