Università degli Studi di Napoli Federico II

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea

Sviluppo di un'applicazione Web per la raccolta e la classificazione, mediante le tecniche dell'Intelligenza Artificiale, di spettri di massa batterici ottenuti con lo spettrometro Maldi-Tof.

Relatore:
Ch.mo Prof. Antonio d'Acierno

Candidato:
Gennaro Cembrola
Matr: 15/19733

Anno Accademico 2004/2005

INDICE

SOMMARIO

Questo lavoro descrive lo sviluppo di un'applicazione Web per l'archiviazione di spettri di massa batterici e per la loro classificazione. Tutti gli spettri di massa trattati sono generati da spettrometri del tipo Maldi-Tof.
In particolare, si illustrano gli studi e le ricerche effettuati per la realizzazione di un sistema di riconoscimento di spettri di massa riguardanti batteri non ancora identificati, mediante tecniche proprie dell'Intelligenza Artificiale. Tale sistema di riconoscimento è stato, poi, sviluppato con il linguaggio Java. Quindi, è stata realizzata un'applicazione Web per l'archiviazione di spettri di massa batterici e per la gestione del relativo archivio e in questa applicazione è stato inglobato il sistema di riconoscimento, rendendo i suoi servizi disponibili sul Web. La gestione dell'archivio permette l'aggiunta, la ricerca e l'eventuale visualizzazione di spettri di massa, mentre il sistema di riconoscimento offre una stima dell'identità degli spettri di massa di batteri non identificati. La parte Web dell'applicazione è stata sviluppata con il framework Jakarta Struts, mentre l'intera applicazione è stata impostata, analizzata e progettata utilizzando i costrutti dell' UML.
Inoltre, in questo lavoro, vengono illustrate le ricerche e gli esperimenti effettuati sui metodi di filtraggio di segnali grezzi e rumorosi, necessari in un sistema di classificazione. Questi metodi, quindi, sono stati adattati agli spettri di massa, che presentano delle marcate peculiarità. Tali peculiarità hanno imposto, inoltre, uno studio ed un confronto tra vari metodi di classificazione applicati agli spettri di massa; vengono esposti, conseguentemente, risultati e considerazioni. Tutto il progetto è risultato molto ambizioso e non è stato completato in tutte le sue parti, richiedendo un tempo superiore al preventivato e degli ulteriori sforzi per completare le ricerche propedeutiche alla sua realizzazione.

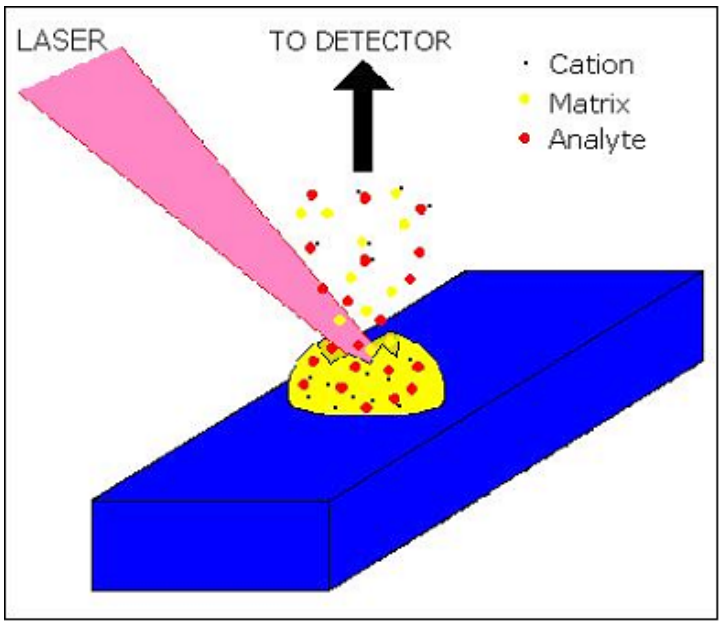
Parte I: Introduzione

Capitolo 1

LO SPETTROMETRO MALDI-TOF E L'INTELLIGENZA ARTIFICIALE

Par 1.1: Lo spettrometro MALDI-ToF

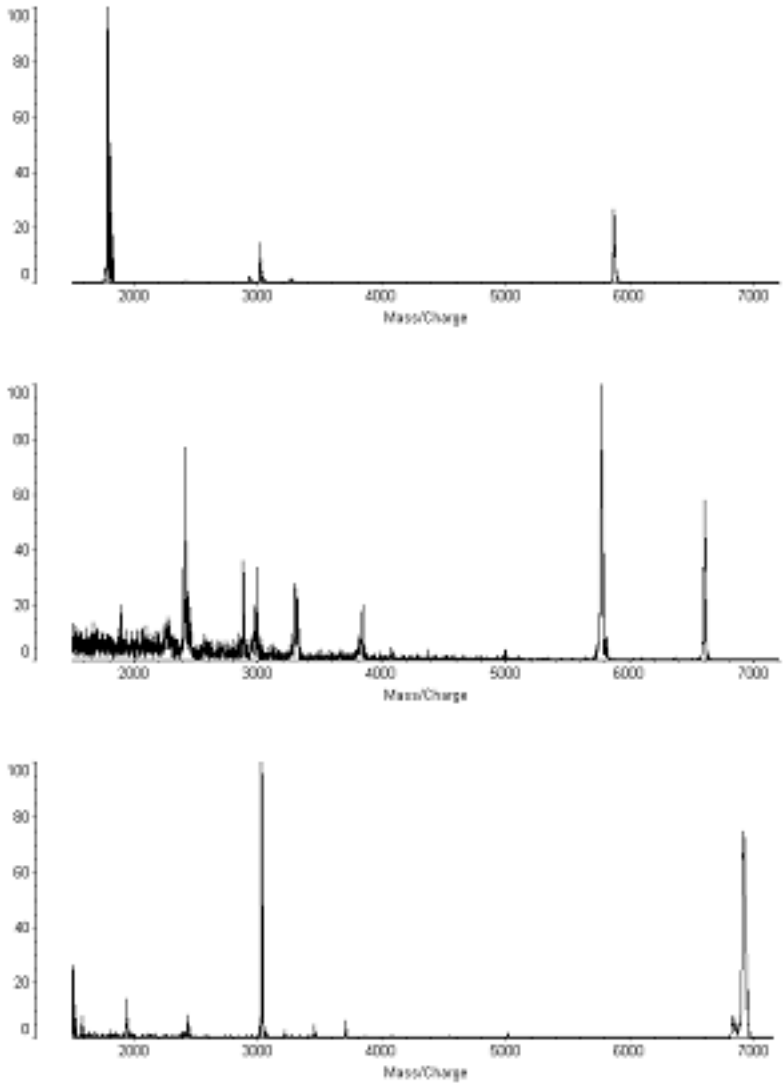
L'identificazione di batteri con i metodi molecolari e biochimici tradizionali è molto lenta e richiede un certo lavoro; nel caso di metodi molecolari si aggiunge anche un certo costo. Un'alternativa a questi metodi è la cosiddetta chemotaxonomy, che teorizza l'individuazione dei batteri attraverso degli identificatori proteici detti markers.
La spettrometria di massa è un potente strumento per realizzare tale metologia.
La spettrometria classica può rilevare masse non oltre i 1000 Dalton, poichè il laser utilizzato sul campione, frammenta le masse più grandi di 1000 Dalton.
La Spettrometria MALDI-ToF (Matrix Assisted Laser Desorption/Ionisation Tine of Flight), supera tale problema utilizzando una matrice per assorbire l'energia del laser ed eliminando, quindi, la possibilità di frammentazione non voluta di masse da rilevare. La Spettrometria MALDI-ToF (Matrix Assisted Laser Desorption/Ionisation Tine of Flight) è stata sviluppata nel 1987 da Hillenkamp, Karas e Tanaka. Questa tecnica usa una biomolecola immersa in un co-precipitato di una matrice che assorbe raggi UV, colpita da un impulso laser di un nanosecondo.



La maggior parte dell'energia del laser è assorbita dalla matrice, mentre la restante parte ionizza parte del campione. La spettrometria si realizza nelle seguenti fasi:

- Le biomolecole ionizzate dall'impulso laser lasciano il campione.
- Tali ioni vengono accelerati in un campo elettrico e convogliati in un tubo (flight tube).
- Durante il volo in questo tubo, le molecole sono separate in funzione del loro rapporto massa-carica e raggiungono il rilevatore in momenti diversi, quindi ogni tipo di molecola genera segnali diversi.

I segnali che si ottengono presentano diversi picchi, che rappresentano le abbondanze degli ioni per un dato rapporto massa-carica:



Utilizzando gli spettri di massa è possibile l'analisi dei batteri isolati che utilizza componenti univoci detti biomarkers per differenziare le specie. Le configurazioni spettrali ottenute da intere cellule batteriche presentano delle cosiddette fingerprints, delle impronte digitali. Le fingerprints sono biomarkers che derivano dalla superficie cellulare cioè dalla membrana delle cellule batteriche. Questi biomarkers sono relativi a proteine immerse in questa membrana. Attraverso queste proteine è teoricamente possibile individuare anche i ceppi batterici, che si differenziano, in gran parte, nell'interazione con l'ambiente, e l'interazione con l'ambiente avviene soprattutto grazie agli elementi immersi nella membrana cellulare.

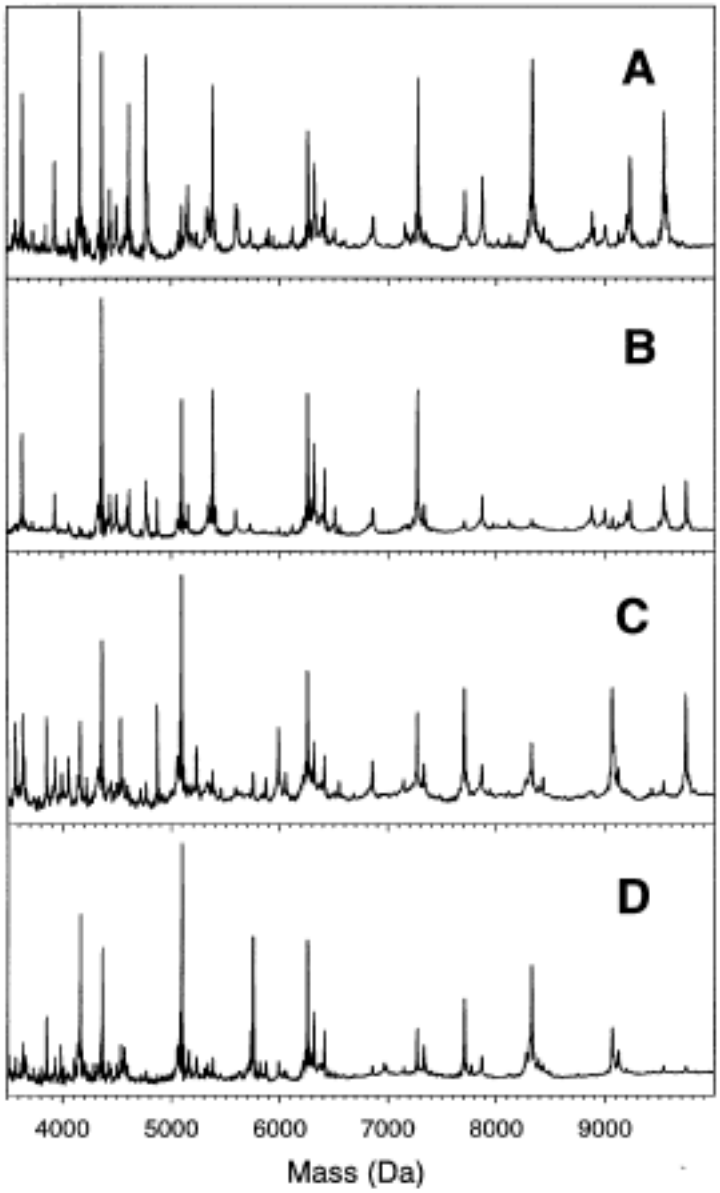


Figure 4. Mass spectra of four different *E. coli* strains, (a) BLR, (b) XL1 blue, (c) RZ1032, and (d) CSH23. Composite correlation index values for these spectra compared to one another are listed in Part A of Table 1.

La dipendenza dall'ambiente può, però, rendere difficile la riproducibilità di un riconoscimento. Se uno stesso batterio viene trattato in modo diverso, con diversi terreni di coltura e in diversi contesti ambientali il suo spettro può variare sensibilmente. Nella seguente figura si evidenzia l'influenza del terreno di coltura sulla riconoscibilità dei batteri.

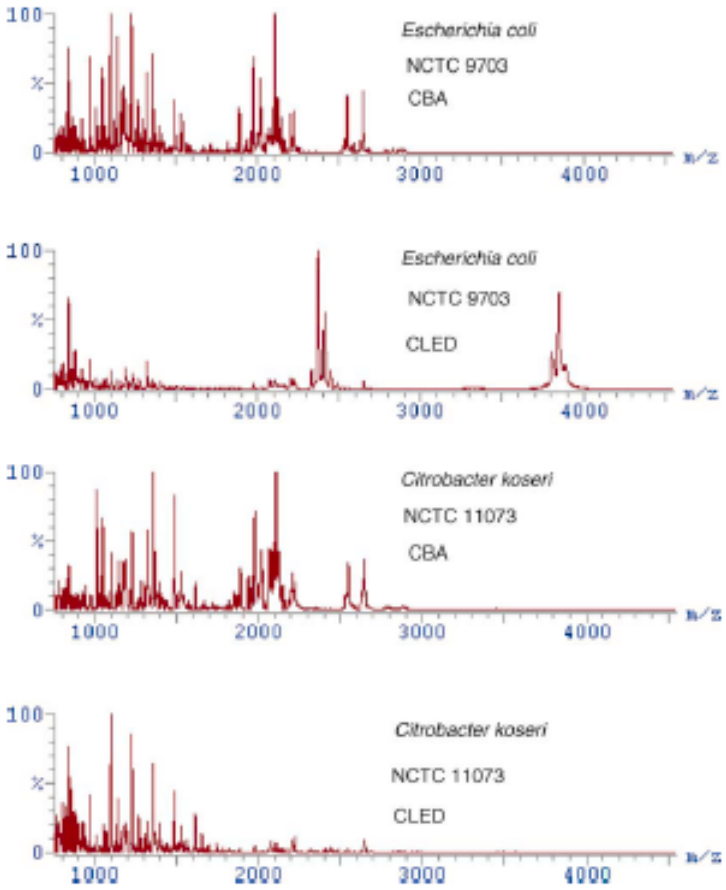


Fig. 8. Comparison of spectral profiles for biochemically similar species of *E. coli* NCTC 9703 and *C. koseri* NCTC 11073 cultured on CBA and CLED agar. Increased differentiation was achieved using CLED agar.

Si nota come, con il variare del terreno di coltura i ceppi batterici possono confondersi.
Quindi si deve realizzare una serie di norme per standardizzare il trattamento dei batteri in modo che siano riconoscibili attraverso la spettrometria.

Par. 1.2: L'Intelligenza Artificiale

Con intelligenza artificiale, AI (dall'inglese Artificial Intelligence), si intende generalmente la possibilità di far svolgere ad un calcolatore alcune funzioni e alcuni ragionamenti tipici della mente umana. Nel suo aspetto puramente informatico, comprende la teoria e la pratica dello sviluppo di algoritmi che rendano le macchine (tipicamente i calcolatori) capaci di mostrare un'abilità e/o attività intelligente anche se in domini molto specifici. L'espressione "Intelligenza Artificiale" (Artificial Intelligence) è stata coniata dal matematico americano John McCarthy in seguito ad uno storico seminario interdisciplinare svoltosi nel New Hampshire nel 1956. Secondo le parole di Marvin Minsky, uno tra i "pionieri" della I.A., lo scopo di questa nuova disciplina sarebbe stato quello di "far fare alle macchine delle cose che richiederebbero l'intelligenza se fossero fatte dagli uomini".
L'approccio dell'IA è quello di trovare una macchina capace non solo di interpretare un algoritmo, ma anche di generarlo ed adattarlo al problema, in netta contrapposizione con la teoria della programmazione classica, dove bisogna conoscere l'algoritmo per risolvere un problema. Prima di iniziare la costruzione di una macchina "intelligente", dobbiamo dare una definizione di "Intelligenza":

- Definizione di Binet: giudicare bene, comprendere bene e ragionare bene.
- Definizione di Spearman: L'intelligenza è la capacità di rapportarsi ad altre macchine.

Questi tentativi di definire l'intelligenza sono astratti e poco concreti. Oggi per l'IA abbiamo definizioni più pratiche basate su approcci ormai consolidati. Una fra le più importanti è quella di Touring, che definisce una macchina "intelligente" se, nascosti un uomo e una macchina dietro un paravento, e considerata un'interfaccia omogenea ed un utente che propone delle domande, l'utente non è in grado di capire se la risposta proviene dall'uomo o dalla macchina. Un altro metodo utilizzato per verificare l'intelligenza è quello dei Test di Bongard. Lo scopo di questi test è trovare due proprietà applicabili a due insiemi di figure (una vera per l'insieme di sinistra e falsa per quello di destra, l'altra vera per quello di destra e falsa per quello di sinistra).
Per risolvere il test parto da una proprietà vera per tutti gli elementi di destra, se questa è vera anche per qualche elemento di sinistra allora è troppo generica e devo particolareggiarla.

Nessuno,in realtà, è ancora riuscito a dare una definizione di intelligenza soddisfacente; proprio a causa di ciò, esistono due principali approcci allo studio dell'AI:

1. la prima, detta tesi forte dell'intelligenza artificiale, sostenuta dai funzionalisti, ritiene che un computer correttamente programmato possa essere veramente dotato di una intelligenza pura, non distinguibile in nessun senso dall'intelligenza umana. L'idea alla base di questa teoria è il concetto che risale al filosofo empirista inglese Thomas Hobbes, il quale sosteneva che ragionare non è nient'altro che calcolare: la mente umana sarebbe dunque il prodotto di un complesso insieme di calcoli eseguiti dal cervello;

2. la seconda, detta tesi debole intelligenza artificiale, sostiene che un computer non sarà mai in grado di essere equivalente a una mente umana, ma potrà solo arrivare a simulare alcuni processi cognitivi umani senza riuscire a riprodurli nella loro totale complessità.

La tesi forte dell'IA riguarda i seguenti punti:

- Simulazione dei meccanismi cognitivi umani
- Nessuna conoscenza pregressa
- Il sistema apprende mediante esempi
- Il sistema opera in due fasi: una di apprendimento e l'altra di decisione.

La tesi debole,invece, stabilisce che:

- Il sistema opera su una base di conoscenza (Knowledge Base)
- La KB è formalizzata da esperti del dominio
- Il sistema risolve tutti e soli i problemi che coinvolgono la conoscenza presente nella KB.

Per quanto concerne il problema della classificazione, in cui si tratta in questo elaborato, esso è parte della tesi forte. In particolare vi sono tre approcci alla classificazione:

a. approccio statistico
b. machine learning
c. reti neurali.

L'approccio statistico riguarda la costruzione di un modello probabilistico, che permette di valutare la probabilità che un campione appartenga ad una determinata classe.
Il machine learning, invece, cerca di costruire delle procedure di decisione automatiche sulla base di operazioni logiche e booleane dopo un apprendimento mediante esempi. In questa teoria hanno molta importanza gli approcci basati su alberi decisionali, ma vi sono anche altri approcci , come quelli che coinvolgono gli algoritmi genetici (GA) e le procedure logice induttive (ILP).
Le reti neurali cercano di imitare la struttura del cervello umano, strutturandosi come un insieme di nodi interconnessi, ognuno dei quali realizza una semplice funzione. I tipi di nodi e l'architettura delle connessioni differenziano le reti neurali. Anche in questo caso la classificazione avviene mediante un apprendimento mediante esempi, ma i meccanismi sono nascosti nella struttura, a differenza dell'approccio statistico.

Par 1.3: La Bioinformatica

La bioinformatica costituisce l'ambizioso tentativo di descrivere dal punto di vista numerico e statistico i fenomeni biologici: storicamente ed epistemologicamente la biologia ha sempre sofferto di una carenza in tal senso rispetto a discipline come la fisica e la chimica, ma oggi la bioinformatica tenta di supplire a questa lacuna fornendo ai risultati tipici della biochimica e della biologia molecolare un corredo di strumenti analitici e numerici davvero promettente. I recenti progressi nel campo della ricerca genomica stanno cambiando il modo di affrontare molti problemi della biologia; di conseguenza la Bioinformatica sta assumendo un ruolo sempre più centrale, sia per quanto riguarda la gestione e l'integrazione dei dati biologici (sequenze di DNA e proteine, strutture proteiche, microarray, proteomica, ecc.), sia per l'elaborazione dell'informazione che molto spesso richiede lo sviluppo di procedure informatiche e algoritmi nuovi. La Bioinformatica è quindi una disciplina in rapida evoluzione. La bioinformatica principalmente si occupa di:

- fornire modelli statistici validi per l'interpretazione dei dati provenienti da esperimenti di biologia molecolare e biochimica al fine di identificare tendenze e leggi numeriche
- generare nuovi modelli e strumenti matematici per l'analisi di sequenze di DNA, RNA e proteine la fine di creare un corpus di conoscenze relative alla frequenza di sequenze rilevanti
- organizzare le conoscenze acquisite a livello globale su genoma e proteoma in basi di dati al fine di rendere tali dati accessibili a tutti, e

ottimizzare gli algoritmi di ricerca dei dati stessi per migliorarne l'accessibilità.

In alcuni casi, il termine bioinformatica viene anche utilizzato a proposito dello sviluppo di algoritmi di tipo informatico che imitano processi di tipo biologico, come ad esempio le **reti neurali** o gli **algoritmi genetici**.

Parte II: L'applicazione Bacteria Classifier

Capitolo 2

INTRODUZIONE

Par. 2.1: Scopo

L'applicazione Bacteria Classifier è un'applicazione WEB con lo scopo principale di coadiuvare l'individuazione di batteri coltivati, sottoposti ad analisi col Maldi-Tof. Inoltre permette la raccolta di spettri di batteri in un database per un' archiviazione razionale e sistematica e per la consulta-zione.
L'applicazione offre una valutazione sull'identità di un batterio attraverso l'elaborazione dello spettro di quest'ultimo e una classificazione. Lo spettro di un batterio sottoposto all'applicazione deve essere trattato per permetterne la classificazione; questo trattamento avviene attraverso diversi stadi ognuno dei quali presenta dei parametri configurabili da parte dell'utente. Inoltre l'utente può scegliere il tipo di classificatore utilizzare.

Par. 2.2: Descrizione

L'applicazione è costituita da due parti principali:

- la parte elaborativa, con gli algoritmi e il database degli spettri batterici
- la parte WEB, responsabile per la disponibilità verso la rete delle sue funzionalità.

La parte elaborativa utilizza vari algoritmi di preprocessing e classificazione che derivano da diversi ambiti tecnico-scientifici. Nel preprocessing, ad esempio, si utilizzano algoritmi per l'elaborazione di segnali come la FFT, la trasformata Wavelet , ecc… Per la classificazione, invece, ci si è avvalsi dei risultati di alcune ricerche nell'ambito dell'intelligenza artificiale e di alcuni algoritmi relativi.
La parte WEB è stata sviluppata ricorrendo al framework Jakarta Struts.

Capitolo 3

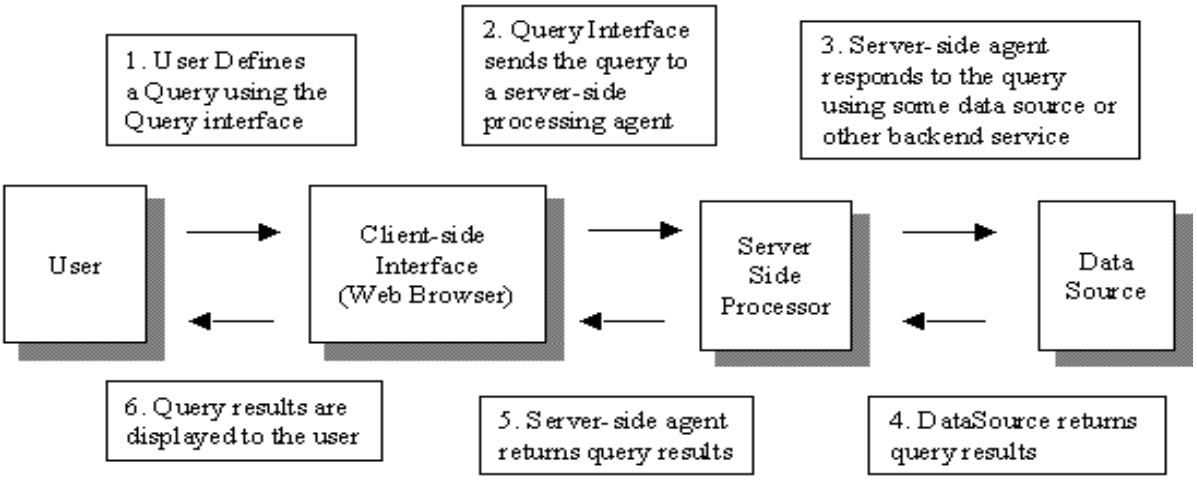
BACTERIA CLASSIFIER: PARTE WEB

Par 3.1: L'applicazione WEB

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 varie 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 richesta 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

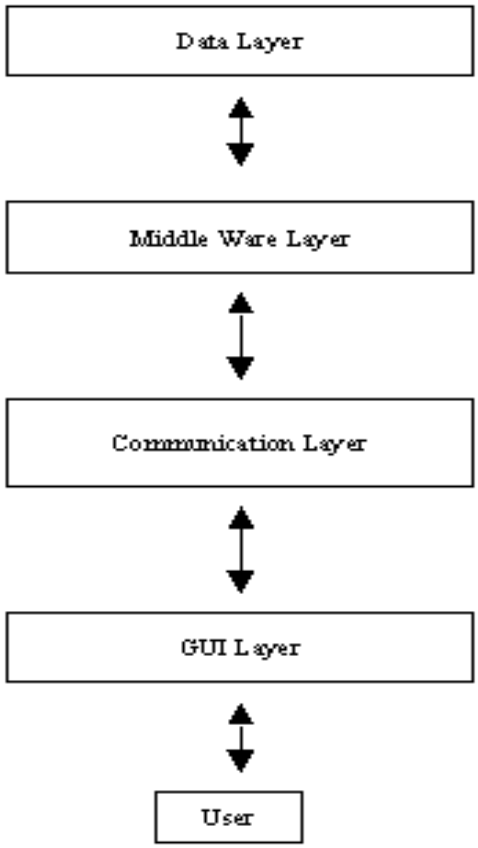
Considerando la seguente figura, si può delineare un tipico flusso di operazioni (work flow) che coinvolge un'applicazione Web:



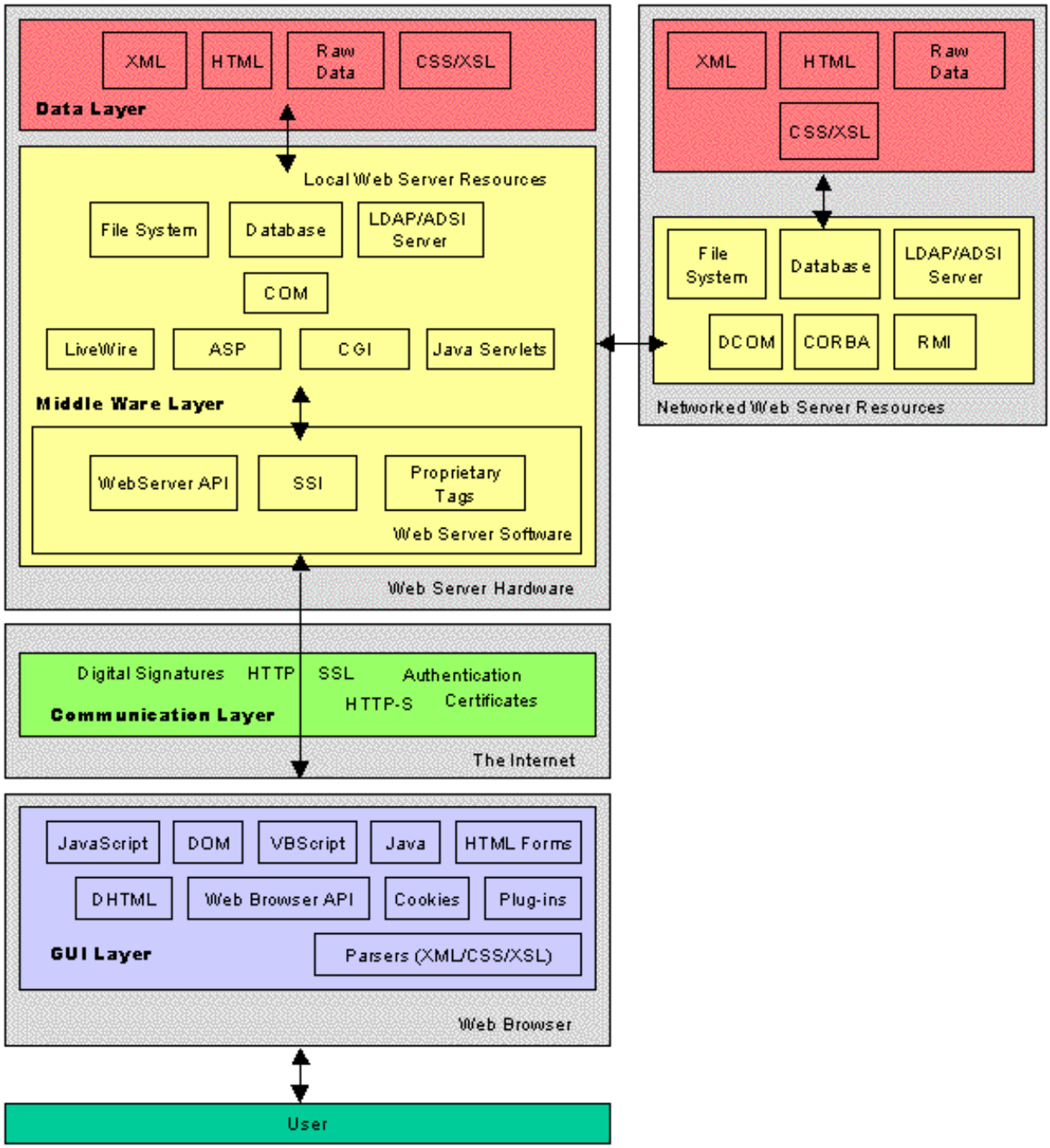
1) un utente effettua una richiesta, magari riempiendo dei campi dell'interfaccia utente;
2) la richiesta con i relativi dati viene inoltrata attraverso la rete al server
3) il server processa la richiesta utilizzando dei dati oppure altri servizi
4) il server raccoglie il risultato della richiesta
5) il risultato viene inoltrato all'utente
6) all'utente viene visualizzato il risultato.

Il fatto essenziale da notare è che per la realizzazione di questo flusso possono esssere utilizzate diverse tecnologie e approcci.
In pratica, vengono utilizzati molteplici strumenti e meccanismi, che combinati in vario modo realizzano questo work flow. 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



Riassumendo le tecnologie disponibili nella loro classificazione si può visualizzare il seguente schema:



Par. 3.2: 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 linguggio 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 quel gap tra uomo e 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 della visualizzazione di informazioni, tipicamente 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.

Par. 3.3: 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.

Par. 3.4: Middle layer

Lo strato intermedio (middle layer) 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.RMI.

Par. 3.5: 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.

Par. 5.6: Java, Servlets, Jakarta Struts

Il layer più importante in un'applicazione Web è il middle ware 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 Bacteria Classifier è stata di utilizzare il linguaggio Java per la realizzazione del middle ware layer..
Questo, in pratica, significa riferirsi all'ambiente J2EE.
Il J2EE (Java 2 Platform Enterprise 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).

Par. 3.7: Il linguaggio 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 nel 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.

Par. 3.8: Servlets

Utilizzare J2EE per il middle ware layer di una applicazione Web significa, in pratica, realizzare dei programmi, che eseguiti sul web server, raccolgono le richieste di un web browser e le soddisfano. Tali programmi sono detti servlets ed i loro compiti sono:

- leggere i dati trasmessi dall'utente
- cercare qualsiasi altra informazione contenuta nella richiesta utente
- generare risultati
- formattare i risultati in un documento
- configurare eventuali parametri della risposta
- mandare il documento all'utente.

Le servlet forniscono pagine di risposta all'utente, ma tali pagine sono create dinamicamente ad ogni richiesta. Questa caratteristica è utile per realizzare la complessa interazione utente-risorsa remota tipica delle applicazioni Web.
Il contenuto dinamico delle pagine di un'applicazione Web può essere dovuto ai dati immessi dall'utente, come nei motori di ricerca, ad informazioni variabili nel tempo come bollettini meteorologici, ad informazioni contenute in

database.Vi sono alcune caratteristiche delle servlets Java che le rendono preferibili ad altre soluzioni del middle ware. Altre soluzioni prevedono l'avvio di un nuovo processo sulla macchina server all'inizio di ogni richiesta e la terminazione dello stesso alla fine di ogni richiesta: per le servlet, eseguite, come ogni programma Java, dalla JVM, si avvia, ad ogni richiesta, un thread.
Il thread ha diversi vantaggi rispetto al processo: è più facile da avviare e terminare; i thread condividono il codice, pertanto ad ogni richiesta vi è un solo codice caricato in memoria; poiché i thread condividono un'area dati, è più facile realizzare ottimizzazioni su molteplici richieste come una connessione al database che può essere condivisa da più utenti. Per certe situazioni, la condivisione di dati tra richieste diverse può essere un problema, ma il Java mette a disposizione meccanismi per realizzare servlets che accedono a dati in modo mutuamente esclusivo. Le servlets, rispetto ad altre soluzioni, presentano una certa facilità d'uso per lo sviluppatore di software, fornendo un 'infrastruttura per la gestione di form HTML, header HTTP, ed altri aspetti peculiari di una applicazione Web. Per certi versi, le servlet utilizzano meglio le risorse, poiché la condivisione di dati riguarda non solo diversi threads di una stessa servlet, ma anche servlets diverse, permettendo certe ottimizzazioni come il pool di connessioni al database. Le servlets sono portabili, e cioè possono essere eseguite su diversi Web server, senza cambiamenti.

Par. 3.9: JSP

L'utilizzo delle servlets Java, implica l'utilizzo dell'infrastruttura Java Server Pages (JSP) per la costruzione di pagine Web di risposta all'utente.
Si potrebbe delegare tutto alle servlets, ma ciò è davvero poco pratico per uno sviluppatore di appplicazioni Web.
Infatti, le JSP permettono di unire contenuto HTML statico a contenuto dinamico generato dalle servlets.
Anche le JSP presentano dei vantaggi che uniti a quelli delle servlets fanno preferire il linguaggio Java come riferimento per lo sviluppo di applicazioni Web.
Infatti, le parti dinamiche delle JSP sono scritte in Java, con tutti i vantaggi che ne derivano, come la portabilità tra piattaforme hardware.
Inoltre introdurre il Java in pagine lato server, significa rendere disponibile tutta la API Java, che facilita certe operazioni come l'accesso ai database.

Par. 3.10: Struts

L'utilizzo delle servlets e delle JSP, oltre i vantaggi visti può presentare degli inconvenienti, soprattuto per quanto riguarda lo sviluppo di applicazioni complesse. Per ovviare a tale inconveniente è stato creato un framework per facilitare la creazione, la gestione, la manutenzione di applicazioni Web J2EE: il framework Jakarta Struts. Struts è un progetto open-source di Apache Jakarta Project , ed è ad oggi il framework largamente più adottato nella comunità degli sviluppatori Java. La sua architettura e i suoi componenti fondamentali presentano pregi e difetti come tutti i framework per gli sviluppatori di software. Un framework è una architettura generica che costituisce l'infrastruttura per lo sviluppo di applicazioni in una determinata area tecnologica: è un insieme di classi ed interfacce di base, che costituiscono l'infrastruttura di una applicazione. In realtà un framework è qualcosa di più complicato di una semplice libreria di classi. Una libreria di classi, quali ad esempio le classi di base del linguaggio Java, viene utilizzata dallo sviluppatore per svolgere determinate funzionalità, ma il flusso di controllo dell' applicazione è delineato dallo sviluppatore stesso: si utilizzano dei pezzi di software già sviluppati per realizzare qualcosa di relativo a specifiche esigenze. E' in pratica un aspetto del paradigma di programmazione ad oggetti che si prefigge, tra le altre cose, di raggiungere una certa modularità nella produzione tipica di certe aree tecniche. Adottare un framework significa invece attenersi ad un specifica architettura, ovvero, nella pratica, estendere le classi del framerwork e/o implementarne le interfacce. In tal caso sono i componenti del framework che hanno la responsabilità di controllare il flusso elaborativo. Nel mondo dell'architettura del software, un framework è considerato come una parte di software esistente nel quale inserire il proprio, ma è il software esistente che invoca e controlla il software aggiunto. Il codice applicativo non è direttamente invocato dall'intervento dell'utente sul sistema ma il flusso elaborativo passa attraverso il codice del framework: sono le classi del framework che invocano codice applicativo e non viceversa come nel caso delle librerie di classi. In definitiva utilizzare un framework significa adottare una specifica architettura per la propria applicazione. Anche se questo può sembrare vincolante è invece , nel caso di un framework valido, uno dei maggiori vantaggi. All'inizio di un progetto si realizzano le scelte fondamentali per la riuscita di un prodotto, ed errori o valutazioni errate in questa fase hanno un costo molto maggiore di errori nella parte più avanzata del processo di sviluppo. La scelta dell'architettura è una delle scelte fondamentali che può determinare il successo o l'insuccesso del progetto stesso. Utilizzare un framework maturo e già ampiamente testato significa attenersi ad una

architettura che funziona e quindi significa iniziare un progetto da una base solida. Ciò porta inoltre ad un significativo risparmio di tempo e risorse in quanto lo sviluppatore non deve più preoccuparsi di realizzare componenti infrastrutturali ma può concentrarsi esclusivamente sullo sviluppo della logica di business che poi è il valore aggiunto della applicazione che si scrive.
Non è raro nello sviluppo di un progetto assistere alla riscrittura di componenti di base che già esistono e che sono stati già ampiamente testati; possiamo dire che uno dei vantaggi nell'utilizzo di un framework è che facilita l'individuazione e l'utilizzo di componenti già realizzati ed affidabili. In definitiva il framework è utile nelle critiche fasi iniziali del ciclo di vita del software, ma anche nelle fasi successive di progetto e implementazione, come nella fase di testing. E' chiaro che tutto ciò è vero quando si fa riferimento ad un framework giunto ad uno stadio di sviluppo maturo, già adottato da molti sviluppatori e quindi già ampiamente provato 'sul campo'. Da un punto di vista pratico adottare un framework significa senz'altro ridurre i tempi di un progetto ed evitare errori nella fase di disegno in quanto si utilizza una infrastruttura realizzata secondo le best-practices dell'ambito tecnologico di riferimento. E' bene precisare che un framework non va confuso con un design-pattern. Un design-pattern è una strategia di soluzione di un problema comune, è qualcosa di concettuale che prescinde dall' implementazione tecnologica. Un framework è invece qualcosa di concreto, è un insieme di componenti che può essere usato per realizzare una applicazione; componenti che, quando il framework è ben strutturato, sono sviluppati secondo i design-pattern più diffusi nell'ambito specifico. Struts implementa molti dei design-pattern J2EE di uso comune , ciò a garanzia di una architettura valida e ben sperimentata. Ovviamente, nell'utilizzo di framework non vi sono solo vantaggi, ma anche svantaggi. In genere i vantaggi dell'utilizzo di un framework vanno ben oltre gli svantaggi, anzi si può affermare che quanto più il progetto sia di grosse dimensioni tanto più l'utilizzo di un framework è altamente consigliabile. E' anche possibile sviluppare un proprio framework , anche se, a meno di casi del tutto particolari, è difficile pensare di scrivere un framework che risolva problematiche diverse da quelle risolte da quelli già esistenti. Di seguito vengono schematicamente riassunti alcuni dei principali vantaggi che si ottengono nell'adozione di un framework nello sviluppo di applicazioni J2EE.

- Disegno architetturale
  Un buon framework è fondato su un disegno architetturale valido, in quanto il suo codice è scritto in base alle best-practices della tecnologia

in uso. Ciò conferisce al proprio progetto fondamenta solide dalle quali partire.

- Riduzione dei tempi di progetto
  Lo sviluppatore deve implementare esclusivamente la logica applicativa potendo risparmiare le energie e il tempo necessari alla scrittura di componenti infrastrutturali.
- 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, custom tags per la presentation logic, internazionalizzazione, validazione dei dati, etc..)

Va precisato che ovviamente un framework non è una panacea o la soluzione di tutti i problemi. Adottarne uno che non si adatta al problema può portare molti svantaggi, per questo la scelta di quello giusto per le specifiche esigenze è di fondamentale importanza. In genere è comunque sempre preferibile evitare framework poco generici, che impongono l'utilizzo di strumenti proprietari e che legano indissolubilmente l' applicazione ad una specifica struttura. Il framework deve fornire una base per lo sviluppo ma la logica applicativa sviluppata deve essere utilizzabile anche al di fuori della struttura del framework stesso, rispettando la modularità del software sviluppato con paradigma ad oggetti.

Esistono molti framework per lo sviluppo di applicazioni web J2EE, sia open-source che prodotti commerciali. I criteri per la scelta di un framework sono molteplici e non esiste il framework 'ideale'. Alcune caratteristiche del framework vanno considerate nella scelta:

- Maturità del progetto
  E' sconsigliabile adottare un framework che sia in una fase iniziale di sviluppo e che sia poco adottato nella comunità degli sviluppatori e quindi poco testato sul campo in progetti reali. Sono preferibili progetti già stabili e sperimentati.
- Documentazione
  Un framework deve fornire documentazione ricca e ben strutturata. Questo facilita la risoluzione dei problemi che si incontrano nella realizzazione dell'applicazione e la comprensione del suo funzionamento.
- Validità del disegno architetturale
  Proprio perché la scelta di un framework influisce sull'architettura applicativa è quindi il framework deve essere disegnato correttamente e

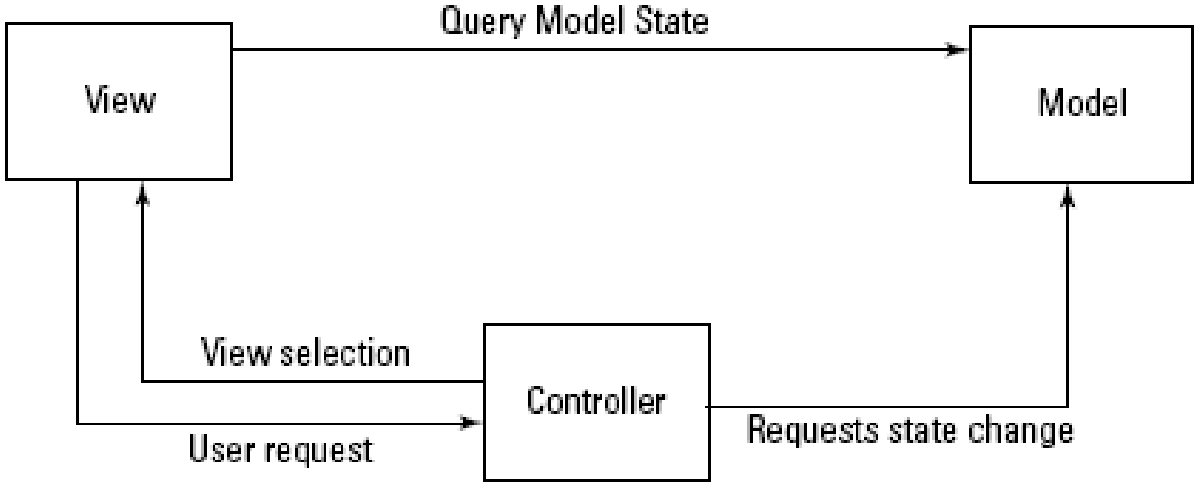
quindi devono esssere stati adottati i design-pattern e le best-practices della tecnologia di riferimento.

- Adozione degli standard
  Un framework deve essere fondato sui componenti standard della tecnologia di riferimento. Quanto più un framework impone soluzioni proprietarie, l'uso di specifici tool di sviluppo o un modello troppo indirizzato ad uno specifico caso applicativo, tanto più è svantaggioso utilizzarlo.
- Estensibilità
  Deve essere possibile estenderne le funzionalità per adattarlo alle specifiche esigenze.

Struts rispetta i criteri sopra elencati e, se utilizzato seguendo alcune principali linee guida, consente di realizzare applicazioni ben strutturate, assolutamente conformi agli standard J2EE e la cui logica applicativa è riutilizzabile anche in altri contesti. Non a caso Jakarta Struts è il framework in assoluto più diffuso a livello mondiale nello sviluppo di applicazioni J2EE.

Par. 3.11: Il pattern MVC (Model-View-Controller)

Jakarta Struts è un MVC web application framework, ovvero è un framework per lo sviluppo di applicazioni web J2EE basato sul pattern Model-View-Controller.



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

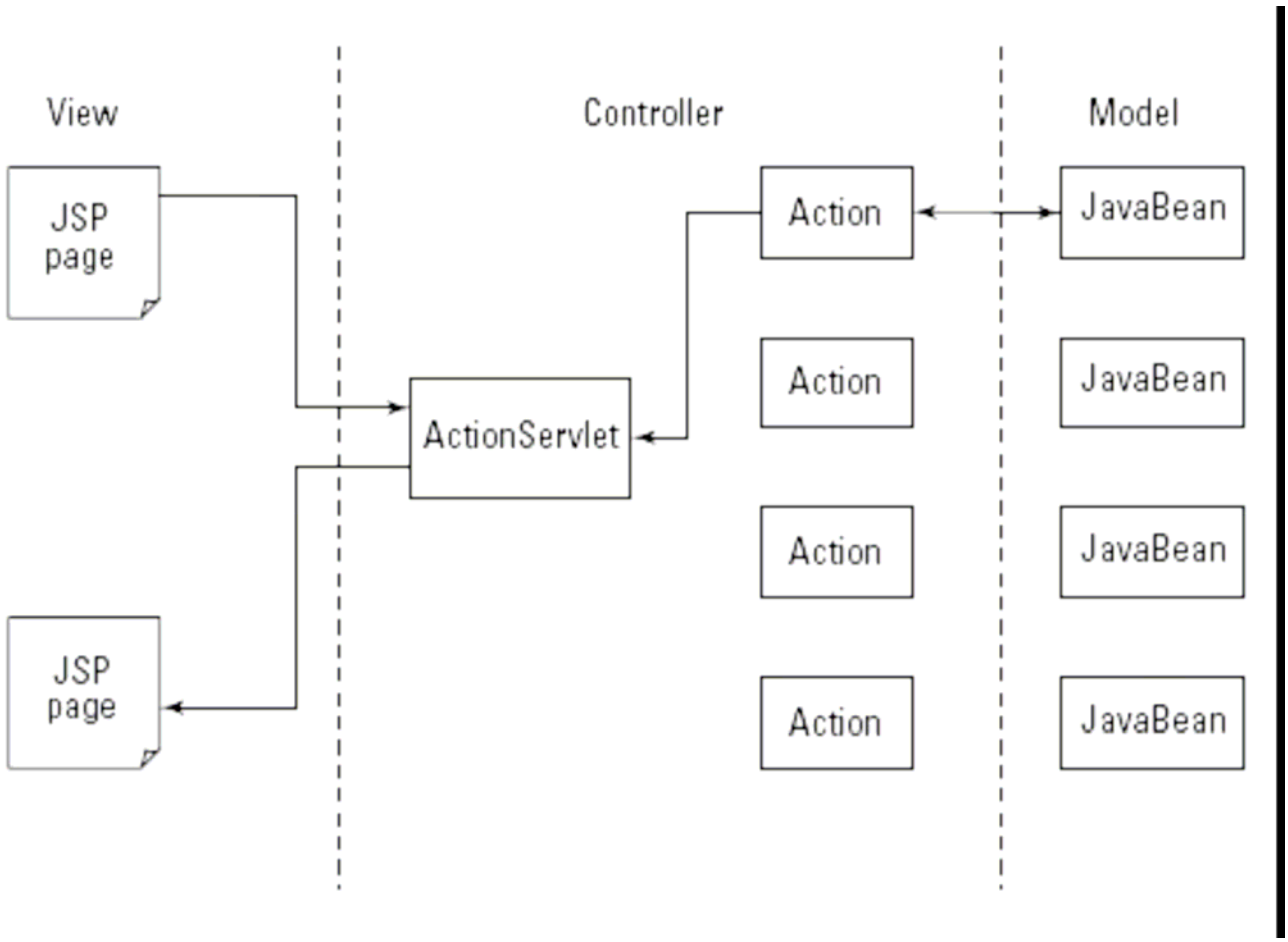
termini di riusabilità e manutenibilità. Esempio tipico di questo problema è l'utilizzo nello sviluppo di una applicazione web J2EE del modello applicativo che nella letteratura è spesso indicato come "JSP Model 1". In base a questo modello l'applicazione è costruita secondo una logica "JSP centric" 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; all'interno delle pagine JSP sono cablati i riferimenti alle viste successive in base alla logica di flusso dell' applicazione che è codificata all'interno della pagina stessa. In questo modello non esiste un controllo centralizzato del flusso, ma ogni vista si fa carico della selezione delle viste ad essa collegate. Un modello di questo tipo , come suggerito dalla stessa Sun, va evitato se non per lo sviluppo di piccoli prototipi o applicazioni molto semplici e dal flusso elaborativo praticamente statico, in quanto porta a scrivere applicazioni difficilmente gestibili al crescere della complessità e non riusabili nei suoi componenti. Quando l'applicazione cresce in complessità non è pensabile svilupparla seguendo un simile approccio. Il pattern MVC è una implementazione di quello che va sotto il nome di "Model 2"; il Model 2 introduce il concetto di controllo centralizzato dell'applicazione, implementato da una servlet di controllo che gestisce tutte le richieste e le soddisfa delegando l'elaborazione a opportune classi Java. In questo modello i ruoli di presentation, control e business logic vengono affidati a componenti diversi e sono tra di loro disaccoppiati, con evidenti vantaggi in termini di riusabilità, manutenibilità, estensibilità e modularità. In un'applicazione costruita secondo il pattern MVC si possono quindi individuare tre livelli logici ben distinti che molto schematicamente svolgono i seguenti compiti:

- *Controller*: 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.
- *Model*: contiene i dati visualizzati dalle viste; è ciò che viene elaborato e successivamente presentato all'utente.
- *View*: visualizza all'utente i dati contenuti nel model. E' la rappresentazione dello stato corrente del Model.

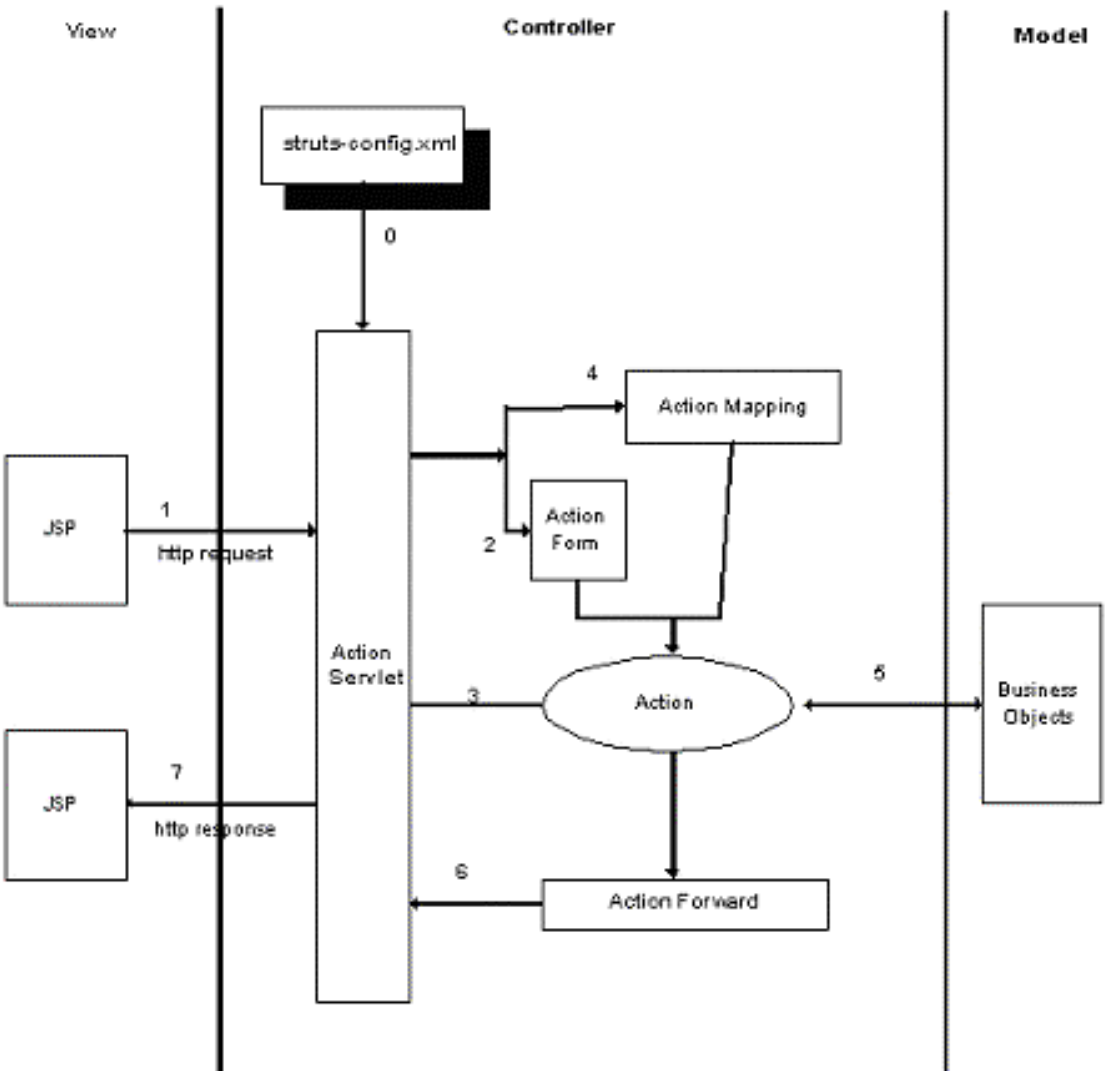
Par 3.12: L'implementazione di Struts del design pattern MVC

Struts è un MVC web application framework, ovvero è un insieme di classi ed interfacce che costituiscono l'infrastruttura per costruire web application J2EE conformi al design pattern MVC. I componenti fondamentali di struts sono:

- *ActionServlet*: e' la servlet di controllo centralizzata che gestisce tutte le richieste dell'applicazione.
- *struts- config.x*: è il file XML di configurazione di tutta l'applicazione. In questo file vengono definiti gli elementi dell'applicazione e le loro associazioni.
- *Action*: le Action sono le classi alle quali la ActionServlet delega l'elaborazione della richiesta.
- *ActionMapping*: contiene gli oggetti associati ad una Action nello struts-config.xml come ad esempio gli ActionForward.
- *ActionForm*: gli ActionForm sono classi contenitori di dati. Vengono popolati automaticamente dal framework con i dati contenuti nelle request http.
- *ActionForward*: contengono i path ai quali la servlet di Struts inoltra il flusso elaborativo in base alla logica dell'applicazione.
- *Custom-tags*: Struts fornisce una serie di librerie di tag per assolvere a molti dei più comuni compiti delle pagine JSP.



La ActionServlet è la servlet di controllo di Struts. Gestisce tutte le richieste client e smista il flusso applicativo in base alla logica configurata. Tutta la configurazione dell'applicazione è contenuta nello struts-config.xml. Questo file XML viene letto in fase di start-up dell'applicazione dalla ActionServlet e definisce le associazioni tra i vari elementi di Struts. Nello struts-config.xml sono ad esempio definite le associazioni tra i path delle richieste http e le classi Action associate alla richieste stesse. Le associazioni tra le Action e gli ActionForm, che vengono automaticamente popolati dal framework con i dati della richiesta ad essi associata e passati in input alla Action. Contiene inoltre l'associazione tra la Action e le ActionForward , ovvero i path configurati nello struts-config.xml ai quali la ActionServlet redirigerà il flusso applicativo al termine della elaborazione della Action.



Nella figura è rappresentato schematicamente il flusso elaborativo nella logica di Struts:

1. Il client invia una richiesta http (1)
2. La richiesta viene ricevuta dalla servlet di Struts che provvede a popolare l'ActionForm associato alla richiesta con i dati della request (2) e l'ActionMapping con gli oggetti relativi alla Action associata alla richiesta (4) . Tutti i dati di configurazione sono stati letti in fase di start-up dell'applicazione (0) dal file XML struts-config.xml.
3. La ActionServlet delega l'elaborazione della richiesta alla Action associata al path della richiesta (3) passandole in input request e response http e l'ActionForm e l'ActionMapping precedentemente valorizzati.

4. La Action si interfaccia con lo strato di business che implementa la logica applicativa. Al termine dell'elaborazione restituisce alla ActionServlet un ActionForward (6) contenente l'informazione del path della vista da fornire all'utente.
5. La ActionServlet esegue il forward alla vista specificata nell'ActionForward (7).

Ovviamente il flusso di operazioni elencato non è completo ma fornisce una indicazione di base su come viene gestito il flusso elaborativo in una applicazione sviluppata con Struts. Anche se il flusso descritto può apparire complesso, è in realtà di comprensione piuttosto semplice.

Par. 3.13: Caratteristiche di base di Jakarta Struts

Vi sono quindi alcune caratteristiche peculiari di Struts, che sono poi comuni anche ad altri MVC framework.

- Esiste una sola servlet di controllo centralizzata. Tutte le richieste sono mappate sulla ActionServlet nel web.xml dell'applicazione. Ciò consente di avere un unico punto di gestione del flusso applicativo e quindi permette di implementare in modo univoco e centralizzato funzioni quali sicurezza, logging, filtri etc.
- Le viste dell'applicazione non contengono al loro interno il riferimento al flusso dell'applicazione e non contengono logica applicativa. I livelli logici dell'applicazione sono disaccoppiati.
- Le viste sono identificate con nomi logici definiti nel file di configurazione struts-config.xml. Nel codice Java non è presente alcun riferimento a nomi di pagine JSP il che rende molto più semplice variare il flusso applicativo.
- Tutta la configurazione dell'applicazione è scritta esternamente in un file XML, il che consente di modificare le associazioni tra le richieste http e le classi ad essa associate in modo molto semplice.

C'è da osservare che tutti i componenti di Struts descritti nel paragrafo precedente fanno parte del livello di controllo tranne i custom-tags che fanno parte della view. Le stesse Action che sono le classi alle quali la ActionServlet delega l'elaborazione delle richieste sono componenti del controller e non del model. Ciò per sottolineare che Struts è un framework model-neutral, ovvero che implementa esclusivamente i livelli di controller e view. Utilizzando Struts è possibile realizzare il livello di business logic in base alle proprie scelte; con

semplici classi Java quindi implementando la logica applicativa nel web-container, o ricorrendo agli EJB quindi sfruttando i servizi del EJB-container.

Par. 3.13: Principali vantaggi nell'uso di Jakarta Struts

Questa discussione introduttiva su Struts non può ovviamente evidenziarne tutti gli aspetti né essere considerata come una trattazione esaustiva del framework, per la quale sarebbe necessario molto più spazio. Da quanto esposto però si possono già evidenziare alcune delle caratteristiche di una applicazione sviluppata con Jakarta Struts e alcuni vantaggi conseguenti al suo utilizzo:

- Modularità e Riusabilità
  I diversi ruoli dell'applicazione sono affidati a diversi componenti. Ciò consente di sviluppare codice modulare e più facilmente riutilizzabile.
- Manutenibilità
  L'applicazione è costiuita da livelli logici ben distinti. Una modifica in uno dei livelli non comporta modifiche negli altri. Ad esempio una modifica ad una pagina JSP non ha impatto sulla logica di controllo o sulla logica di business, cosa che avveniva nel JSP Model 1.
- Rapidità di sviluppo
  A differenza di quanto avveniva utilizzando il JSP Model 1, è possibile sviluppare in parallelo le varie parti dell'applicazione, view (JSP/HTML) e logica di business (Java) sfruttando al meglio le conoscenze dei componenti del team di sviluppo. Si possono utilizzare sviluppatori meno esperti e anche con poche conoscenze di Java per la realizzazione delle view, permettendo agli sviluppatori Java più esperti di concentrarsi sulla realizzazione della business logic.
- Tag HTML
  Struts fornisce una collezione di custom tag utili nella creazione della view di un'applicazione, che ne facilitano lo sviluppo e il debug.

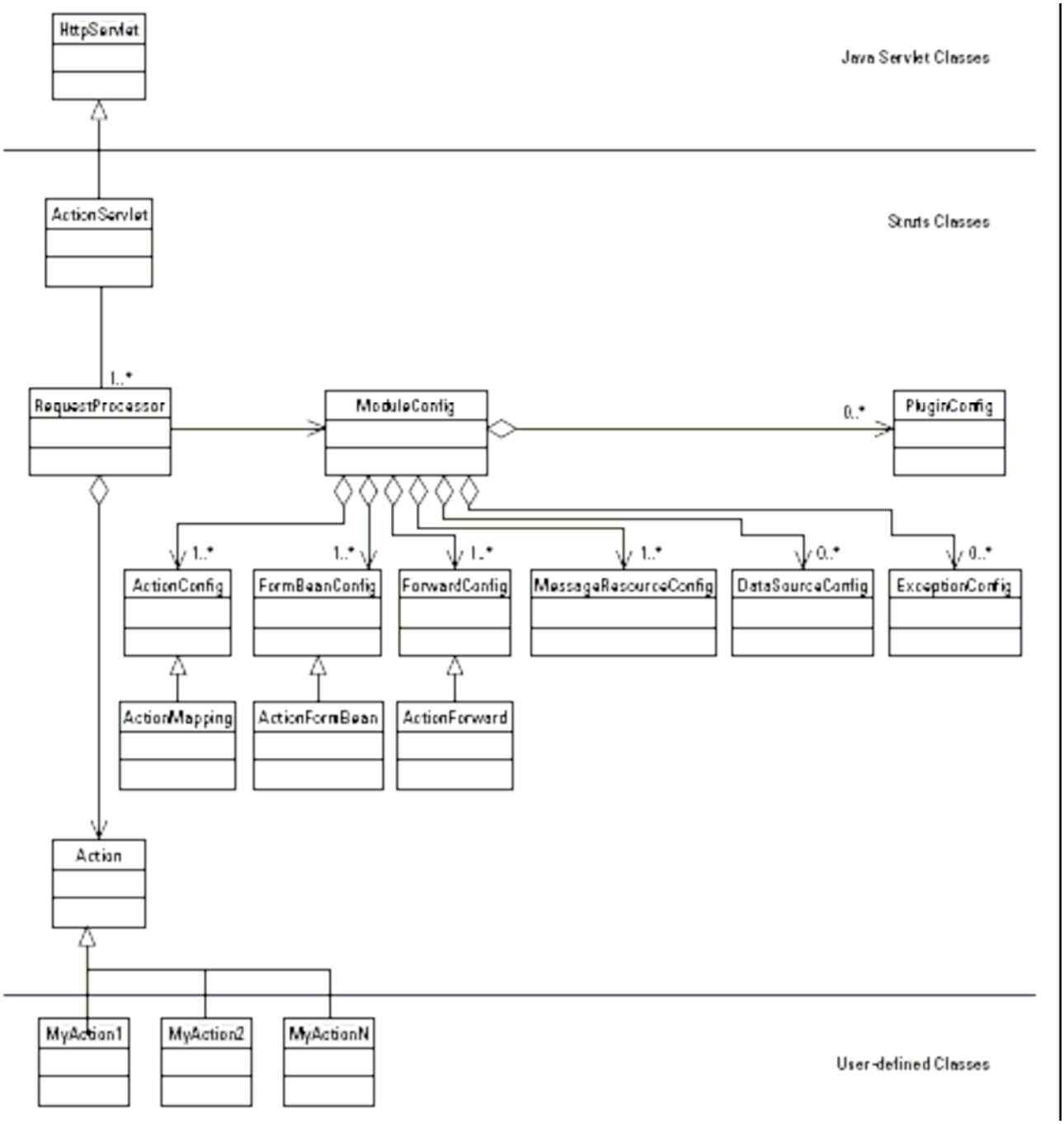
Par. 3.14: Principali svantaggi nell'uso di Jakarta Struts

- Curva di apprendimento ripida
  Il framework Struts è abbastanza ampio ed elaborato.Questo implica uno sforzo non trascurabile per gli sviluppatori neofiti nell'apprendere tutti gli strumenti messi a disposizione dal framework. Inoltre se il

progetto è di piccole dimensioni il framework risulta troppo corposo per esigenze limitate. Quindi in questi casi è meglio ricorrere ad altri framework MVC, se non addirittura rinunciare all' MVC per qualcosa di meno strutturato, ma sicuramente più immediato

- Documentazione
  Confrontata con la documentazione delle API servlets e JSP, la documentazione dello Struts è più confusa e meno dettagliata e organizzata.
- Poco trasparente
  Struts nasconde una parte della dinamica dell'applicazione e quindi applicazioni sviluppate con Struts sono più difficili da capire, ma soprattutto sono molto complicate da ottimizzare, almeno per quanto riguarda la parte Web.

Par. 3.15: Le classi di controllo Struts



La ActionServlet

La org.apache.struts.action.ActionServlet è la servlet di controllo di Struts. E' la servlet che gestisce tutte le richieste http che provengono dai client e indirizza il flusso applicativo in base alla configurazione presente nel file XML struts-config.xml.

Come è ovvio la ActionServlet estende la javax.servlet.http.HttpServlet; i suoi metodi doGet() e doPost() chiamano entrambi un metodo process() che esegue quindi l'elaborazione sia in caso di richieste di tipo GET che di tipo POST.
La ActionServlet esegue l'elaborazione che schematicamente comprende i seguenti step:

1. I metodi doGet() e doPost() invocano il metodo process() della ActionServlet
2. Nel metodo process() la ActionServlet ottiene l'istanza del RequestProcessor, configurato per l'applicazione nel tag <controller> dello struts.config.xml, e ne esegue il metodo process().
3. Nel metodo process() del RequestProcessor viene eseguita l'elaborazione vera e propria, ed in output al metodo viene fornito un oggetto ActionForward che consente alla ActionServlet di inoltrare l'elaborazione in base alla configurazione presente nello struts-config.xml.

La ActionServlet viene configurata, come ogni servlet, nel web.xml. I parametri di inizializzazione sono molti e per un elenco completo si rimanda al sito ufficiale di Struts (http://jakarta.apache.org/struts/). Di seguito è riportato un blocco <servlet> di configurazione standard della ActionServlet nel quale è settato ad 1 il parametro <load-on-startup> in base al quale il container istanzia la ActionServlet allo start-up della web-application e ne invoca il metodo init(). Inoltre mediante il parametro config è specificata la posizione del file XML di configurazione dell'applicazione. Il parametro debug abilita il debugging dell'applicazione. Mediante il blocco <servlet-mapping> si specifica che tutte le richieste con path terminante in .do vengono mappate sulla servlet di controllo di Struts:

```
<!-Configurazione standard della Action Servlet -->
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
<init-param>
<param-name>config</param-name>
<param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
<init-param>
<param-name>debug</param-name>
<param-value>2</param-value>
</init-param>
```

```
<load-on-startup>1</load-on-startup>
</servlet>

<!- Mapping della Action Servlet -->
<servlet-mapping>
<servlet-name>action</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Il Request Processor

La org.apache.struts.action.RequestProcessor è la classe alla quale la ActionServlet delega l'elaborazione delle richieste. Il RequestProcessor viene configurato mediante il tag <controller> dello struts-config.xml , ed è possibile utilizzarne uno proprio scrivendo una classe che estende la org.apache.struts.action.RequestProcessor e ne implementa i metodi. In particolare è di uso comune fare l'override del metodo processPreprocess() che viene eseguito dal RequestProcessor prima dell'elaborazione di ogni richiesta. Questo metodo è il punto ottimale per inserire controlli di validità della sessione, dell'utente o simili.

Il RequestProcessor esegue i seguenti step:

1. Legge il file struts-config.xml per trovate un un elemento XML <action> corrispondente al path della richiesta.
2. Una volta trovato l'elemento <action> corrispondente verifica se è presente l'attributo *name* che corrisponde al nome dell'ActionForm configurato per la richiesta in elaborazione. In tal caso provvede a reperire una istanza dell'ActionForm e a popolarne gli attributi con i valori presenti nella request http, facendo una corrispondenza tra nome parametro e nome attributo.
3. Se nell'elemento <action> è presente l'attributo *validate* al valore true chiama il metodo validate() dell'ActionForm per il controllo dei dati forniti dalla request.
4. Se il controllo è ok a questo punto il RequestProcessor chiama il metodo execute() dell'Action configurata nell'elemento <action> delegandole l'elaborazione della richiesta.
5. Il metodo execute() dell'Action al termine dell'elaborazione restituisce un oggetto ActionForward che consente al RequestProcessor di inoltrare il flusso elaborativo.

Come visto quindi la ActionServlet delega l'elaborazione della richiesta al RequestProcessor che a sua volta , dopo aver popolato con i dati della request l'ActionForm configurato nell'elemento <action> corrispondente al path della

richiesta, delega l'elaborazione della singola richiesta alla classe Action corrispondente. Un esempio di configurazione di una Action nel file struts-config.xml è il seguente:

```
<!-definizione del ActionForm -->
<form-beans>
<form-bean name="startForm" type="it.prove..MenuForm" />
</form-beans>

<action-mappings>
<!-definizione del Action -->
<action path="/start"
name="startForm"
scope="request"
type="it.prove.StartAction"
validate="true">
<forward name="ok" path="/pagina1.jsp"/>
<forward name="ko" path="/errorpage.jsp"/>
</action>
</action-mappings>
```

In questo esempio al path /start.do viene associato il form it.prove.StartForm e la Action it.prove.StartAction. Ciò significa che quando il RequestProcessor riceverà una richiesta con path /start.do valorizzerà gli attributi di un oggetto della classe it.prove.StartForm con i parametri della request e dopo averne validato i valori la passerà al metodo execute() della classe it.prove.StartAction che esegue l'elaborazione prevista.

La classe Action

La classe Action è l'elemento fondamentale del controller di Struts in quanto per ogni funzione realizzata con Struts bisogna creare una propria classe che la estende e ne implementa il metodo execute() che è fatto come segue:

```
public ActionForward execute(ActionMapping mapping, ActionForm
form,  HttpServletRequest request, HttpServletResponse response)
                throws Exception
{

  //codice di esempio
```

```
  //acquisizione form
  MyForm myForm = (MyForm)form;
  //acquisizione parametri dal form
  String param1 = myForm.getParam1();

  //business logic
  ...
  //fine business logic
  //inoltro dell'elaborazione
  return mapping.findForward("ok");
}
```

Il metodo execute() riceve in input request e response http, un'istanza dell'oggetto ActionForm prima descritto, e un oggetto ActionMapping che contiene le infornazioni configurate nell'elemento <action> tra le quali i forward, ovvero i percorsi a cui inoltrare in uscita l'elaborazione. Restituisce un oggetto ActionForward che contiene il path di inoltro dell'elaborazione. E' nel metodo execute() della propria Action che lo sviluppatore inserisce il proprio codice di elaborazione della richiesta per la funzione specifica. Bisogna subito sottolineare due aspetti fondamentali riguardo alle Action:

1. Le Action vengono gestite esattamente come delle servlet. Ovvero il loro funzionamento è basato sulla stessa logica multithread delle servlet quindi sono soggette a tutti i problemi comunemente noti nelle servlet. Il codice scritto nelle Action deve essere thread-safe per un corretto funzionamento delle stesse.
2. Le Action fanno parte del Controller e non del Model. La logica applicativa non deve essere scritta nella Action, ma questa deve delegare allo strato di Model l'elaborazione della business-logic.

In base a quanto detto una Action dovrebbe:

a. Acquisire i dati della request dal form
b. Delegare l'elaborazione della business-logic alle classi del Model
c. Acquisire i risultati dell'elaborazione e prepararli per la vista da inviare all'utente mettendoli nello scope opportuno (se necessario).
d. Inoltrare il flusso elaborativo in base alla logica applicativa.

Le Action costiuiscono quindi il 'ponte' applicativo tra lo strato di Controller e di Model di un aapplicazione scritta con Struts ed hanno un ruolo fondamentale perché sono le classi che lo sviluppatore scrive continuamente nello sviluppo di una applicazione Struts.

La classe ActionForm

Abbiamo già visto che i form (come sono chiamate nella terminologia Struts le classi che estendono la org.apache.struts.action.ActionForm) sono sostanzialmente dei bean contenenti dati, che il framework popola automaticamente con i dati della request svincolando lo sviluppatore dal doverlo fare con proprio codice applicativo. Associando ad un path un oggetto di una classe che estende la org.apache.struts.action.ActionForm, definito mediante il tag <form-bean>, è possibile fornire al metodo execute() della Action un oggetto contenente tutti i dati inseriti in un FORM HTML, con la possibilità di validarli prima che gli stessi giungano alla Action stessa come visto in precedenza. In ogni applicazione web, la view ha due compiti fondamentali: presentare all'utente i dati frutto dell'elaborazione eseguita e consentire all'utente l'immissione di dati da elaborare. Normalmente in una applicazione J2EE tradizionale i dati da visualizzare sono contenuti negli attributi di un JavaBean memorizzato nell'appropriato scope al quale la pagina fa riferimento. L'immissione di dati è realizzata mediante un FORM Html contenuto nella pagina JSP e i vari input type che consentono l'invio nella request di dati che saranno reperiti dai componenti del controller e passati allo strato di business logic. Il reperimento dei dati dalla request, la loro validazione e il popolamento con essi degli oggetti del model è a carico dello sviluppatore che dovrà scrivere codice per reperire i dati dalla request , istanziare un oggetto di una classe opportuna per memorizzarli , validarli e fornirli allo strato di business-logic.Gli ActionForm di Struts consentono di automatizzare in parte questo compito che è uno dei più frequenti e ripetitivi in una applicazione web J2EE. Un ActionForm è una classe che estende la org.apache.struts.actions.ActionForm e che viene utilizzata per acquisire i dati da un form HTML e fornirli ad una classe Action. In pratica il controller di Struts provvede a popolare in automatico gli attributi di un ActionForm associato ad una determinata Action con i dati inviati nella request, associandoli in base alla corrispondenza nome-parametro nome-attributo, e a passare l'istanza dell'ActionForm così valorizzata al metodo execute() della Action stessa. Gli ActionForm costiuiscono quindi una sorta di buffer nel quale vengono posti i dati digitati in un form HTML, che possono così essere ripresentati facilmente all'utente in caso di errori nella validazione. Allo stesso tempo costituisce un filtro per l'applicazione in quanto facilita il controllo dei dati prima che questi vengano passati allo strato di logica.
Gli ActionForm sono del tutto equivalenti a dei JavaBean e possono essere quindi utilizzati per contenere i dati restituiti dallo strato di business logic e da presentare all'utente oltre che a essere usati come classi corrispondenti ad un form Html come il loro nome suggerisce. Di seguito è riportato il codice di

esempio di un ActionForm corrispondente ad un classico form di immissione di username e password:

```
public class LoginForm extends ActionForm {
  private String password = null;
  private String username = null;
  public String getPassword() {
    return this.password;
  }
  public void setPassword(String password) {
    this.password = password;
  }
  public String getUsername() {
    return this.username;
  }
  public void setUsername(String username) {
    this.username = username;
  }
}
```

Come si vede la struttura è esattamente quella di un JavaBean, a parte l'estensione della classe ActionForm. Gli ActionForm estendono la org.apache.struts.actions.ActionForm , ed hanno attributi privati e corrispondenti metodi get e set pubblici. In più hanno due metodi particolari: reset() e validate() che ne caratterizzano il comportamento..
Il metodo reset() viene chiamato dal controller dopo che questo ha creato o reperito dallo scope opportuno l'istanza dell'ActionForm. Può quindi essere usato per inizializzare gli attributi del form ad un valore stabilito.
Il metodo validate() viene chiamato dal controller dopo la valorizzazione degli attributi dell'ActionForm qualora nello struts-config.xml sia stato valorizzato a true l'attributo validate del tag <action> nel quale si fa riferimento all'ActionForm in questione. Nel metodo validate() va inserito il codice per la validazione formale dei dati del form. Ciò garantisce di avere un punto standard nel codice nel quale questa validazione viene effettuata e che i dati che arrivano alla Action siano già stati formalmente validati.

Gli ActionErrors

Il metodo validate() di un ActionForm è il punto nel quale viene inserito il codice di validazione formale dei dati immessi dall'utente in un form HTML. La signature del metodo è la seguente:

```
public ActionErrors validate( ActionMappings mapping,
HttpServletRequest request)
```

Il tipo di ritorno del metodo è un oggetto della classe ActionErrors che è un contenitore di oggetti della classe org.apache.struts.action.ActionError. Ogni oggetto della classe ActionError rappresenta un errore verificatosi nella validazione dei dati. Qualora durante la validazione si verifichino degli errori, per ciascuno di essi viene creata una istanza di un oggetto ActionError e aggiunta all'oggetto ActionErrors restituito dal metodo. Se il controller verifica che l'oggetto ActionErrors in uscita al metodo validate() non è nullo, non trasferisce il controllo al metodo execute() della classe Action associata alla richiesta in elaborazione ma bensì alla pagina JSP il cui path è configurato nell'attributo input del tag <action> corrispondente.Con un opportuno custom tag (<html:errors>) posto nella pagina stessa sarà possibile visualizzare i messaggi di errore associati agli errori verificatisi senza scrittura di codice aggiuntivo. Il messaggio di errore viene reperito automaticamente dal framework dal resouce bundle dell'applicazione; la chiave del messaggio è fornita nel costruttore dell'oggetto ActionError associato all'errore in questione. Di seguito è riportato l'esempio di un metodo che esegue la validazione della username e della password immessi nel form html associato all'ActionForm visto in precedenza:

```
public ActionErrors validate(ActionMapping
mapping,HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();
  if ((username == null) || (username.length() < 1))
    errors.add ("username",new
ActionError("errore.username.obbligatorio"));
  if ((password == null) || (password.length() < 1))
    errors.add("password",new
ActionError("errore.password.obbligatoria"));
  return errors;
}
```

Le label errore.username.obbligatorio e errore.password.obbligatorio sono le chiavi alle quali sono associati i messaggi di errore nel resource bundle dell'applicazione.Con il metodo validate, le classi ActionErrors ed ActionError ed il tag <html:errors> il framework fornisce quindi un automatismo standard

per la gestione della validazione dei dati immessi nella view dell'applicazione e per la visualizzazione dei messaggi di errore.

Le librerie di custom-tags di Struts

Per la costruzione delle viste dell'applicazione , Struts mette a disposizione alcune librerie di tag che svolgono alcuni dei compiti più frequenti in una pagina JSP. Le librerie di tag sono raggruppate logicamente in base al tipo di funzione svolta e sono:

- html Tag, per la generazione di form HTML che interagiscono con gli ActionForm e degli altri elementi HTML di una pagina JSP
- bean Tag, usati per accedere a proprietà di JavaBeans e per creare istanze di bean
- logic Tag, usati per logica condizionale, iterazioni e controllo di flusso
- nested Tag, che estendono le funzionalità dei tag base.

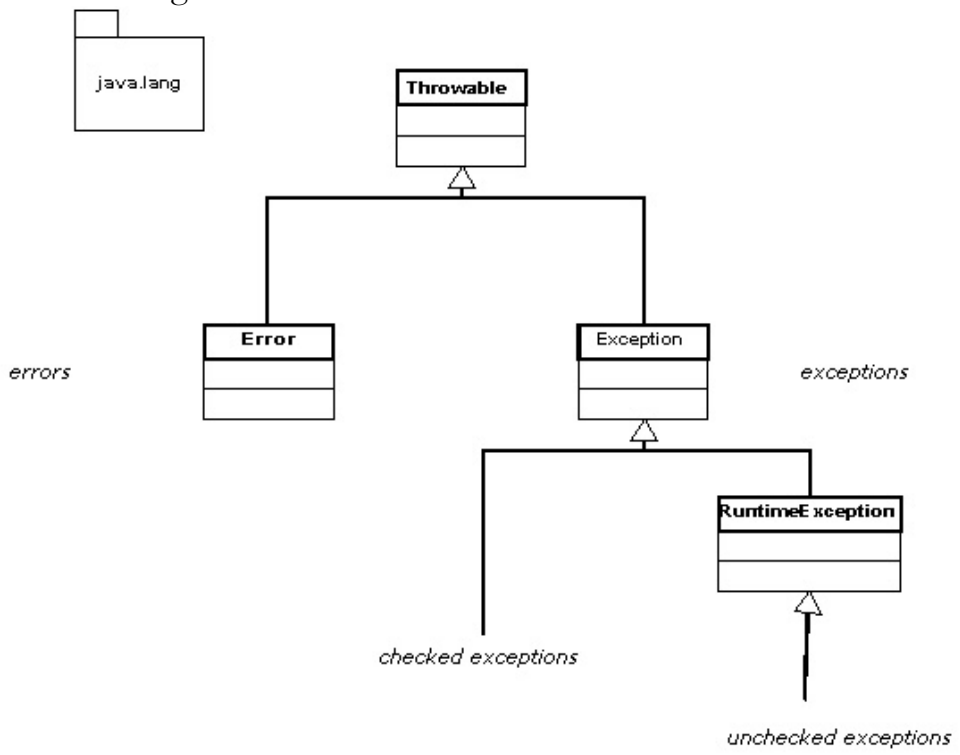
La libreria html comprende una serie di tag che consentono di generare in automatico ed in maniera standard tag html. In particolare per ciò che riguarda i form HTML questi tag sono strettamente agli ActionForm.
Il tag <html:form> è senza dubbio uno dei più comuni in quanto consente di definire un form HTML. Vi sono poi tag per ciascuno degli input type html.
Nella libreria logic esistono numerosi tag per eseguire logica condizionale quali <logic:empty> <logic:notEmpty> <logic:equal> <logic:notEqual> <logic:greaterThan> ed altri., tag per eseguire iterazioni su array e collection quali <logic:iterate>, e tag per eseguire controllo di flusso come il <logic:redirect>. Nella libreria bean sono presenti tag per la definizione di variabili di scripting utilizzabili all'interno della pagina quali <bean:define> , tag per la scrittura in output del valore di attributi di un <bean: write> e così via. Va precisato che l'utilizzo dei custom tag nelle pagine JSP è altamente consigliato per evitare il proliferare di scriptlet Java nelle pagine e per rendere standard la scrittura delle stesse. E' molto importante inoltre osservare che alcuni dei tag forniti nelle librerie di Struts si sovrappongono per la loro funzione con i tag presenti nelle JSTL (Java Standard Tag Library) introdotte di recente nella J2EE. In particolare i tag della libreria logic di Struts trovano corrispondenza in molti dei tag della libreria core delle JSTL. L'indicazione è di usare sempre questi ultimi in quanto sono già lo standard per la scrittura delle pagine JSP e sicuramente verranno utilizzati nelle versioni future.
Probabilmente la libreria di tag di Struts che avrà ancora largo utilizzo è la html almeno finchè la tecnologia delle Java Server Faces non si sarà diffusa ed affermata.

I componenti di Struts per la gestione dell'internazionalizzazione

Struts fornisce supporto all'internazionalizzazione essenzialmente per ciò che riguarda il reperimento di testo e immagini localizzate. Per gli altri aspetti, quali formattazione di date, importi etc. bisogna fare ricorso alle classi Java standard. Struts gestisce l'internazionalizzazione, fornendo gli strumenti per reperire risorse localizzate da opportuni resource bundle in base al locale corrente.

Le eccezioni in Java

Le eccezioni sono il meccanismo che il linguaggio Java mette a disposizione per gestire eventuali situazioni anomale che possono verificarsi nell'esecuzione di una applicazione. Una eccezione è un oggetto di una determinata classe che viene creato quando una certa situazione di errore si è verificata in conseguenza dell'esecuzione di una applicazione. Le eccezioni fanno parte di una gerarchia di classi che ha come padre la classe java.lang.Throwable come rappresentato in figura.



Da essa discendono due classi fondamentali che sono la java.lang.Exception e java.lang.Error. Le classi che discendono da Error rappresentano situazioni anomale dalle quali non è possibile in genere ripristinare l'applicazione, quali un OutOfMemory o uno StackOverflowError. Le classi che ereditano da Exception, e alle quali facciamo riferimento come eccezioni, invece

rappresentano quelle situazioni anomale che lo sviluppatore può intercettare e gestire in base alla logica della propria applicazione. Le eccezioni si suddividono ulteriormente in checked exceptions ed unchecked exceptions. La suddivisione fa riferimento all'obbligo da parte dello sviluppatore di dover gestire in qualche modo una eccezione o meno. Le unchecked exceptions sono sottoclassi di java.lang.RuntimeException che a sua volta è una sottoclasse di java.lang.Exception. Le eccezioni che discendono da RuntimeException possono essere gestite allo stesso modo di quelle checked ma non vi è l'obbligo di farlo, cioè il compilatore non impone che le eccezioni che estendono la RuntimeException vengano gestite. La gestione delle eccezioni a livello di linguaggio viene fatta mediante l'uso del blocco try/catch e della clausola throws. Il codice che potenzialmente può generare eccezioni può essere messo all'interno di un blocco try/catch . Qualora si verifichi una eccezione il flusso elaborativo passa all'interno del blocco catch corrispondente all'eccezione generata, all'interno del quale viene messo il codice per la gestione dell'eccezione stessa, e prosegue con il codice seguente il blocco try/catch. Per le eccezioni di tipo checked questa gestione è obbligatoria, ovvero il compilatore non compilerà con successo la classe se il codice che può generare eccezione non è inserito all'interno di un blocco try/catch , a meno che il metodo stesso non dichiari che la gestione della eccezione è delegata al metodo chiamante. In tal caso si utilizza nella definizione del metodo la clausola throws. Mediante la clausola throws si dichiara quali eccezioni possono essere generate dal metodo in questione informando il chiamante sui propri obblighi in tale senso, tutto secondo la logica del design by contract . Nel caso in cui anche il metodo chiamante dichiari nella clausola throws l'eccezione in questione, l'elaborazione passerà all'ulteriore metodo chiamante e così via fino alla fine dello stack contenente la sequenza dei metodi.

Struts e la gestione delle eccezioni: approccio programmatico o dichiarativo

Dalla versione 1.1 di Struts sono state introdotte nel framework delle componenti che consentono di gestire le eccezioni in modo strutturato ed efficiente. I due approcci possibili nella gestione delle situazioni di eccezione, così come nella gestione di molte altre problematiche applicative ricorrenti nello sviluppo di applicazioni J2EE, sono quello programmatico e quello dichiarativo. In generale l'approccio programmatico consiste nello scrivere all'interno della propria applicazione codice specifico per la gestione di una determinata problematica mentre l'approccio dichiarativo consiste nel configurare all'esterno dell'applicazione le modalità di gestione del problema stesso. Nel caso delle eccezioni quindi, gestire una situazione di eccezione in

modo programmatico significa inserire all'interno del codice delle istruzioni specifiche per il trattamento dell'eccezione verificatasi. L'approccio dichiarativo invece presuppone la configurazione all'esterno del codice del comportamento dell'applicazione a fronte del verificarsi di una eccezione. Il primo approccio è quello tradizionale , utilizzabile in una qualsiasi applicazione Java e non presenta particolari differenze in una applicazione sviluppata con Struts. In questo caso lo sviluppatore deve inserire all'interno delle proprie classi il codice opportuno, ovvero i blocchi try/catch per la gestione delle eccezioni, né più né meno che in una qualsiasi applicazione Java. Una Action contenente gestione programmatica delle eccezioni potrebbe avere un aspetto simile:

```
.
.
try{
  //codice che può generare eccezioni
}
catch(Exception e){
  ActionErrors errors = new ActionErrors();
  erorrs.add("eccezione", new ActionErorrs("eccezione"));
  saveErrors(request,errors);
  return mapping.findForward("eccezione");
}
```

In pratica si tratta di racchiudere il codice tra blocchi try/cath e nel catch corrispondente mettere il codice di gestione dell'eccezione che nell'esempio non è altro che la generazione di un errore da inviare al client, secondo la modalità già nota. Questo approccio è possibile ed è sicuramente migliorabile personalizzando ad hoc il framework con classi generalizzate che accentrino gran parte della gestione necessaria, ma costringe comunque lo sviluppatore a scrivere codice ad hoc, spesso ridondante, per gestire queste situazioni. Inoltre in questo caso se si vuole modificare il comportamento dell'applicazione al verificarsi di una eccezione è necessario intervenire nel codice. E' invece sicuramente più elegante ed efficiente sfruttare la possibilità offerta da Struts di gestire in modo dichiarativo le eccezioni. All'esterno del codice , quindi nel file XML di configurazione struts-config.xml, è possibile descrivere mediante opportune sezioni XML il comportamento che l'applicazione deve avere a fronte di una particolare eccezione. In questo modo, non solo si evita di scrivere codice specifico nell'applicazione per la gestione delle eccezioni, ma si ha la possibilità di modificare il comportamento dell'applicazione senza intervenire nel codice stesso.

Gli strumenti forniti da Struts per la gestione dichiarativa delle eccezioni

Il principio di base è quello di definire esternamente al codice il comportamento dell'applicazione e il conseguente flusso applicativo al verificarsi di una eccezione. Questa configurazione può essere fatta a livello di singola Action o a livello globale per tutta l'applicazione.
Per configurare la gestione a livello di singola action si utilizza il tag <exception> all'interno della configurazione della Action stessa. Nel tag <exception> si dichiara la classe dell'eccezione da gestire, la risorsa a cui verrà inoltrato il flusso elaborativo ed una chiave per acquisire dal resource bundle dell'applicazione un messaggio di errore come nell'esempio seguente:

```
<action path"/esempio"
  type="it.esempi.EsempioAction"
  input="input.jsp"
  scope="request"
  name="myForm" >
  <exception key="label.eccezione.esempio"
    type="it.esempi.MiaException"
    path="errorpage.jsp" />
</action>
```

In questo esempio si definisce che a seguito di una ipotetica MiaException il flusso applicativo sarà inoltrato alla pagina JSP errorpage.jsp acquisendo dal resource bundle il messaggio di errore corrispondente alla label label.eccezione.esempio. In questo modo è possibile configurare tutti i comportamenti in corrispondenza delle possibili eccezioni generate dal metodo execute() di una Action. E' anche possibile definire un comportamento globale utilizzando il tag <global-exception>. In tal caso si presuppone la definizione di una o più classi handler per le eccezioni da gestire, ovvero delle classi che estendono org.apache.struts.action.ExceptionHandler ed il cui metodo execute() viene eseguito dal framework al verificarsi della corrispondente eccezione. La configurazione di una eccezione a livello globale è fatta come nell'esempio seguente:

```
<global-exceptions>
  <exception key="label.eccezione"
    type="it.esempi.MiaException"
```

```
    handler="it.esempi.ExceptionHandler"/>
</global-exceptions>
```

Nella sezione <global-exception> si dichiara la classe dell'eccezione da gestire, la classe handler che conterrà la logica di gestione ed una chiave per acquisire dal resource bundle dell'applicazione un messaggio di errore. Nell'esempio riportato si definisce che la classe it.esempi.ExceptionHandler , che estende org.apache.struts.action. ExceptionHandler, gestisce le eccezioni di tipo it.esempi.MiaException. Ciò significa che al verificarsi di una eccezione di questo tipo il framework manderà in esecuzione il metodo execute() della classe handler così dichiarata. In questo metodo in genere è opportuno inserire del codice per effettuare il log dell'eccezione, memorizzando tutte le informazioni utili alla comprensione del problema che la ha generata, quali il messaggio associato all'eccezione, lo stack-trace etc. Il metodo restituisce in output un oggetto della classe ActionForward che consente di effettuare il forward alla risorsa desiderata.E' ovvio che in entrambi i casi non sarà necessario inserire all'interno dell'applicazione alcun codice di gestione delle eccezioni perché il tutto è delegato alle classi handler.

Il Validator

Il Validator è un framework che fornisce gli strumenti per effettuare in modo automatico e dichiarativo i controlli formali sui campi digitati in un form HTML.Usando il Validator non è necessario scrivere alcun codice di validazione nel metodo validate() degli ActionForm, ma è il Validator stesso che fornisce questa funzione purchè i form bean dell'applicazione estendano uno degli ActionForm del Validator stesso. Il Validator è costiuito da un insieme di classi predisposte per eseguire tutti i più comuni controlli di validazione in genere usati nelle applicazioni, ma.esiste anche la possibilità di creare routine di validazione non fornite dal Validator. Il Validator inoltre supporta sia la validazione server-side che quella client-side mediante opportune funzioni JavaScript, cosa non fornita dal meccanismo standard degli ActionForm di Struts.

Le Action base standard di Struts

Struts fornisce alcune Action che arricchiscono il framework di alcune funzionalità rispetto alla Action base standard e che sono utili a diversi scopi . Segue una descrizione di quelle più utilizzate nella pratica.

La DispatchAction

Nel funzionamento standard di Struts , il framework esegue il metodo execute() della Action che corrisponde al URL della richiesta effettuata dal client. Il metodo execute() costituisce quindi il punto di ingresso nel framework a fronte di una richiesta dal client. Questo funzionamento si adatta poco alle situazioni nelle quali è necessario eseguire una serie di elaborazioni tra loro logicamente collegate. Tipico è l'esempio di operazioni di inserimento, cancellazione , lettura e aggiornamento su una stessa tabella di un database. Sarebbe abbastanza poco efficiente dover definire una Action per ciascuna singola operazione , in quanto questa tecnica porterebbe ad un proliferare di Action nell'applicazione e quindi ad una difficile gestione della stessa. Struts ci viene incontro fornendo org.apache.struts.actions.DispatchAction.
La DispatchAction è assolutamente analoga ad una Action base ma fornisce la possibilità di invocare diversi metodi della stessa purchè il client specifichi il metodo da chiamare. In pratica è come una Action che non ha un solo metodo execute() ma ne ha n con nomi diversi. Ognuno di questi metodi deve avere la stessa signature del metodo execute(). Affinché il framework sappia a quale metodo delegare l'elaborazione della richiesta, il client deve fornire nella request un parametro contenente il nome del metodo corrispondente. Questo parametro va ovviamente specificato nella definizione della Action nello struts-config.xml. Se quindi da una pagina JSP si vuole invocare un metodo inserisci() della Action GestioneTabellaAction sarà sufficiente specificare:

http://servername/context-root/gestioneTabella?metodo=inserisci

Chiaramente il nome del metodo può essere specificato in diversi modi, come un hidden contenuto in un form HTML oppure può essere impostato da una funzione JavaScript prima di eseguire la submit() del form.
L'importante è che è possibile raggruppare logicamente azioni tra di loro correlate in un'unica Action il che porta ad una migliore strutturazione dell'applicazione ed evita duplicazioni inutili di codice.

La LookupDispatchAction

La org.apache.struts.actions.LookupDispatchAction è utile quando la scelta del metodo da eseguire in una Action di tipo 'dispatch' è effettuata mediante i button di un form ma si ha la necessità di avere le label dei button localizzate e non ci si vuole affidare a codice JavaScript per la selezione del metodo da attivare.La LookupDiapatchAction è quindi del tutto analoga alla DispatchAction per quel che riguarda la sua struttura, quello che cambia è il modo con il quale viene selezionato il metodo da mandare in esecuzione. In

questo caso le label dei button vengono associate alle loro key contenute nel resource bundle dell'applicazione, e queste key, che molto probabilmente non sono nomi di metodo validi, vengono mappate dallo sviluppatore ai metodi della Action mediante la definizione di un metodo così fatto:

```
protected Map getKeyMethodMap(ActionMapping mapping,
ActionForm form,
                        HttpServletRequest request) {
  Map map = new HashMap();
  map.put("bottone.leggi", "leggi");
  map.put("bottone.inserisci", "inserisci");
  map.put("bottone.cacnella", "cancella");
  map.put("bottone.modifica", "modifica");
  return map;
}
```

Nel metodo viene definita una HashMap nella quale ad ogni key è associato il nome di un metodo della Action. Questo metodo è usato dal framework per determinare la corrispondenza tra la label localizzata del bottone cliccato ed il metodo della LookupDispatchAction da eseguire.
In base a questo codice la definizione dei bottoni nella pagina JSP sarà del seguente tipo:

```
<html:form action="/gestioneTabella">
<html:submit property="method">
<bean:message key=" bottone.leggi ">
</html:submit>
<html:submit property="method">
<bean:message key=" bottone.inserisci ">
</html:submit>
<html:submit property="method">
<bean:message key=" bottone.cacnella ">
</html:submit>
<html:submit property="method">
<bean:message key=" bottone.modifica ">
</html:submit>
</html:form>
```

In questo modo è possibile avere una Action di tipo 'dispatch' i cui metodi

sono attivabili da button di un form di una applicazione localizzata semplicemente scrivendo un metodo getKeyMethodMap() come descritto.

Le Action standard di Struts per il controllo del flusso

Struts fornisce alcune standard che consentono di controllare il flusso elaborativo.

La ForwardAction

La ForwardAction consente di inoltrare il flusso dell'applicazione ad un'altra risorsa individuata mediante un URI valido, risorsa che può essere una pagina JSP una servlet o altro. La ForwardAction quindi non fa altro che creare un RequestDispatcher ed effettuare il forward alla risorsa individuata dall'attributo parameter specificato nella definizione della Action nello struts-config.xml.

```
<action
path="/vaiAllaPagina1"
type="org.apache.struts.actions.ForwardAction"
name="pagina1Form"
scope="request"
input="/pagina0.jsp"
parameter="/pagina1.jsp "/>
```

La ForwardAction è molto utile quando nell'applicazione si hanno pagine JSP che non richiedono alcuna elaborazione a monte prima di essere visualizzate. Affinché si eviti di effettuare un inoltro alla pagina in questione direttamente da un'altra pagina JSP dell'applicazione si può usare la ForwardAction. In questo modo si resta aderenti al modello di Struts che prevede un controllo centralizzato di tutte le richieste e si predispone l'applicazione a modifiche future. Se un domani infatti la pagina in questione richiedesse invece qualche elaborazione basterà sostituire una Action opportuna al mapping precedentemente corrispondente alla ForwardAction. Un altro utilizzo della ForwardAction è come elemento di integrazione con altre applicazioni data la sua caratteristica di effettuare un inoltro ad un URI generico.

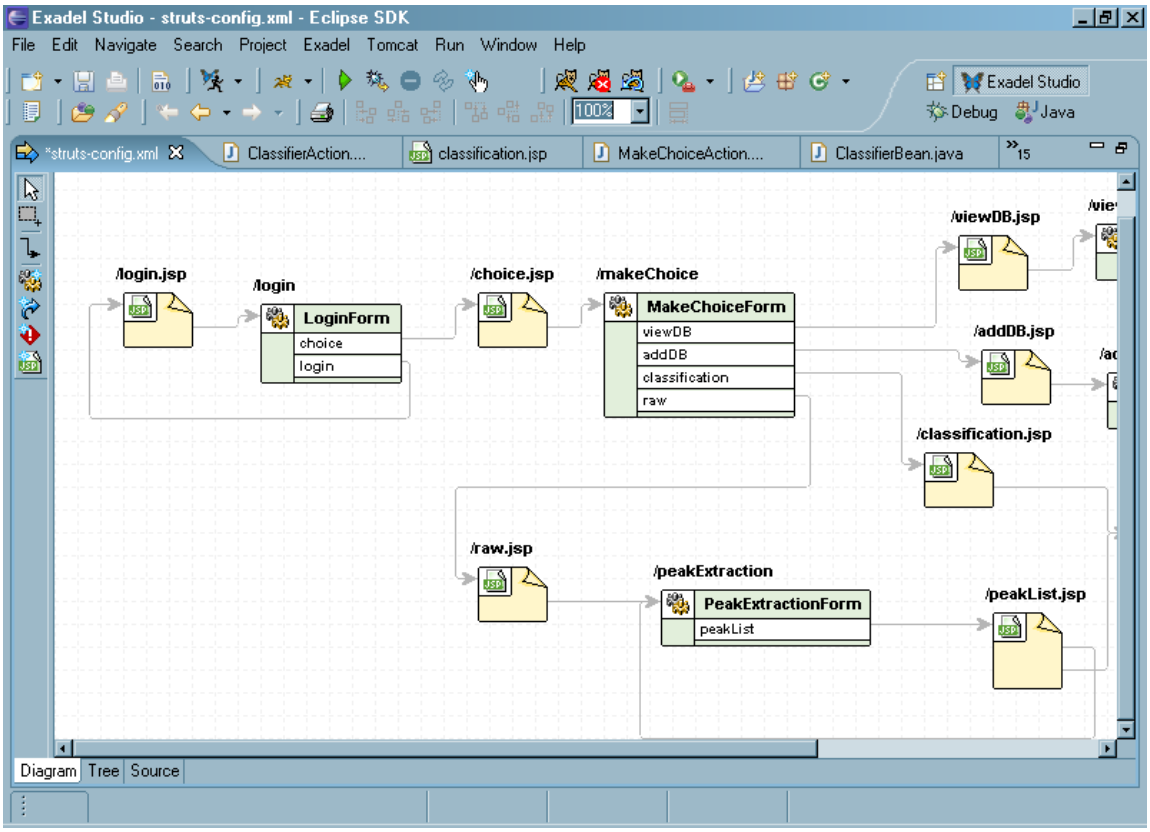
La IncludeAction

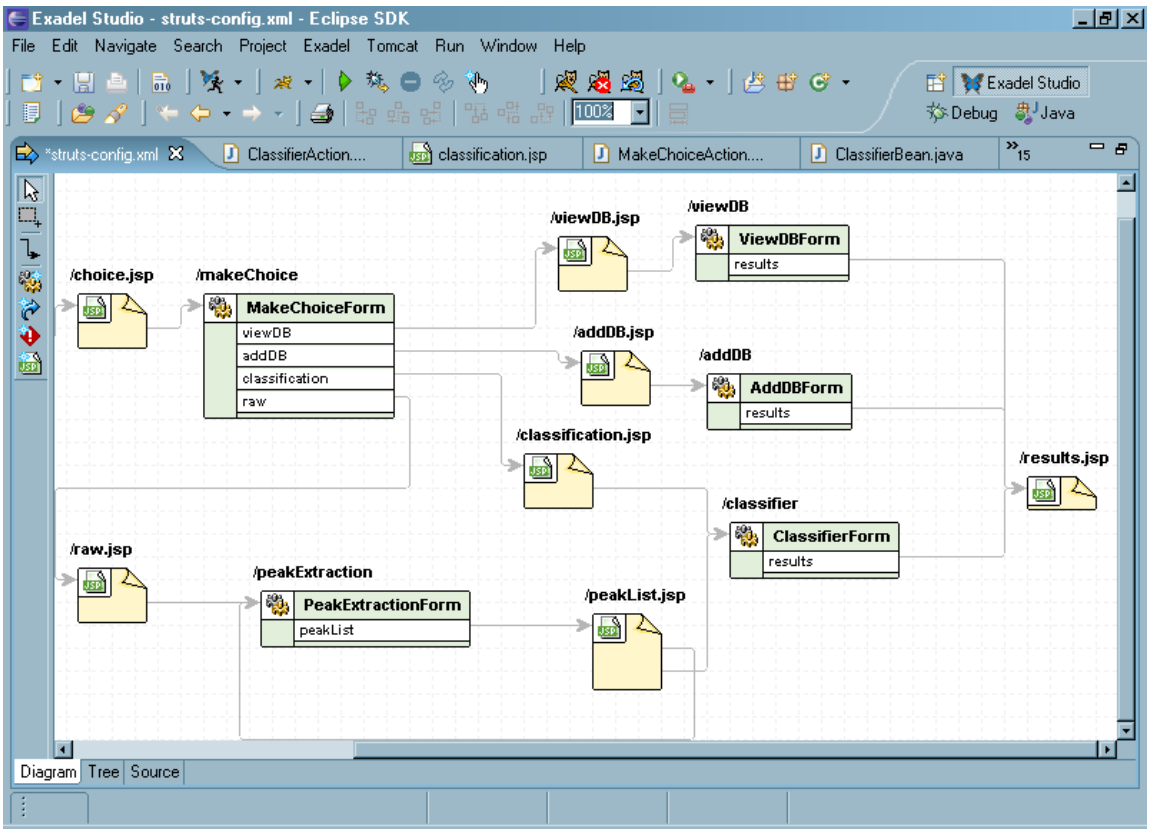
La IncludeAction è la corrispettiva della ForwardAction per l'operazione di include della risposta generata da un'altra risorsa. Consente di aggiungere alla propria elaborazione l'elaborazione effettuata da un'altra risorsa quale ad

esempio una servlet. Non è molto utilizzata ma è comunque fornita da Struts qualora potesse servire.

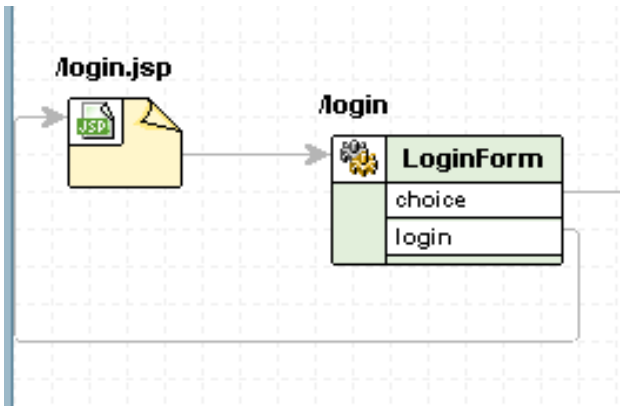
Par. 3.16: Descrizione della parte Web dell'applicazione

L'applicazione Bacteria Classifier è stata sviluppata utilizzando il framework Jakarta Struts. Seguendo il modello MVC, tutto ciò che fa parte dell'interazione con l'utente e con la rete Web è stato realizzato mediante gli strumenti messi a disposizione da Struts. La logica applicativa ed il database sono stati concepiti a parte seguendo una certa modularità, pensando a questi elementi come indipendenti dal contesto Web.
Nella parte Web sono stati utilizzati le Action, le tags, i meccanismi di controllo degli ingressi, i form beans ed altre tecniche messe a disposizione da Struts. Seguono due immagini della struttura del file struts-config.xml, come visualizzata dall'exadel(poiché la struttura è ampia, la sua schermata è stata divisa a metà, per chiarezza):

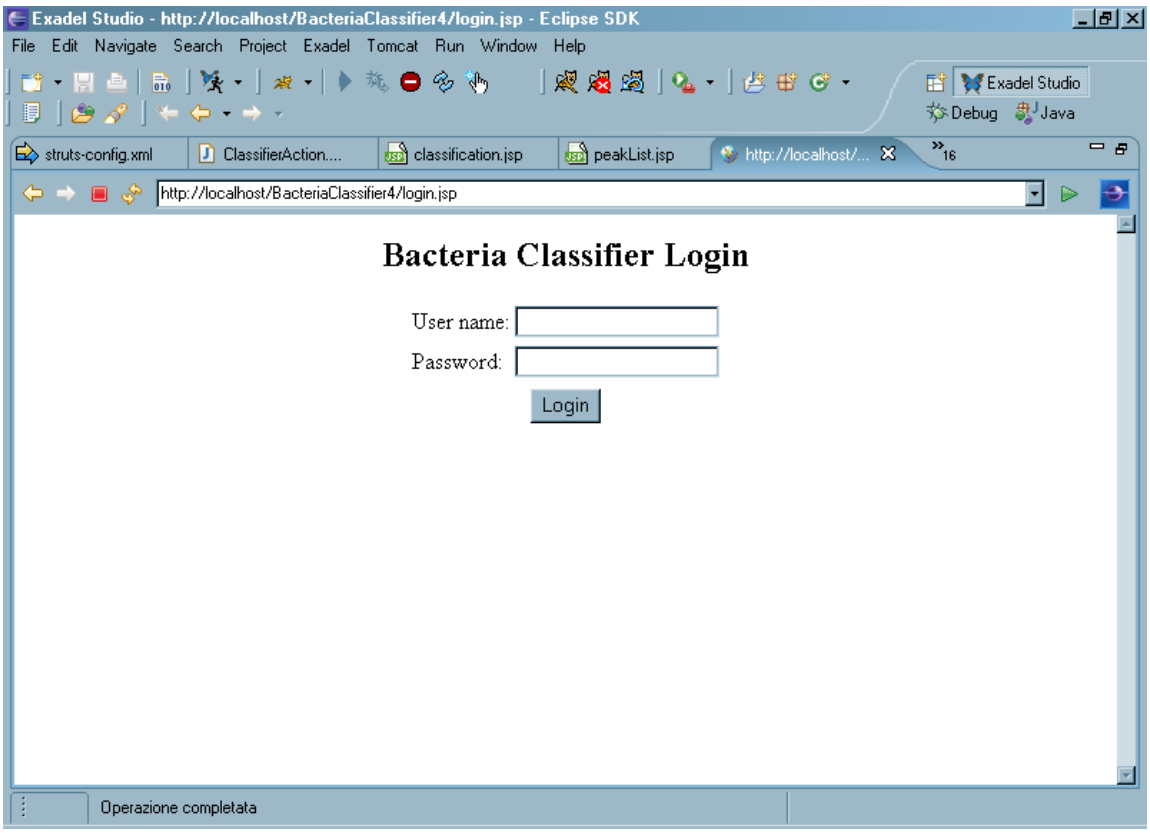




Par. 3.17: Login



La pagina login è il punto d'ingresso ai servizi dell'applicazione ed è anche il punto in cui si pùo notare l'utilizzo di importanti elementi di Struts.



Nella pagina login vi è un form per l'immissione dei dati di utente, user name e password, che vanno verificati per garantire l'accesso all'applicazione.
In questo form, va controllata, innanzitutto, la digitazione dei campi. Struts mette a disposizione un meccanismo per questo controllo, che permette di valutare la digitazione di un campo prima che i valori di questi raggiungano la action di controllo relativa al form. Mediante un metodo 'validate', codificato nel form bean che gestisce il form, si può verificare che i campi siano almeno stati digitati non appena il form viene trasmesso. Controlli di questo tipo, sono in questo modo, estratti dalla logica della action servlet, che risulta più alleggerita. La action può risultare il bottle-neck di un' applicazione Web, essendo coinvolta in tutte le richieste che arrivano al ramo dell'applicazione a cui appartiene. Vi è un gran vantaggio prestazionale nello spostare il controllo di condizioni che non coinvolgono la logica applicativa e ne il database server, nel form bean che raccoglie i dati di un form: è inutile coinvolgere la action, la logica applicativa, le risorse del server, quando alcune configurazioni di ingresso sono non valide, indipendentemente dal Model. Quando il *RequestProcessor* riempie con i dati del form il relativo bean, chiama automaticamente, se configurato in *struts-config.xml*, il metodo validate del form bean, che immediatamente verifica alcune condizioni sugli ingressi.

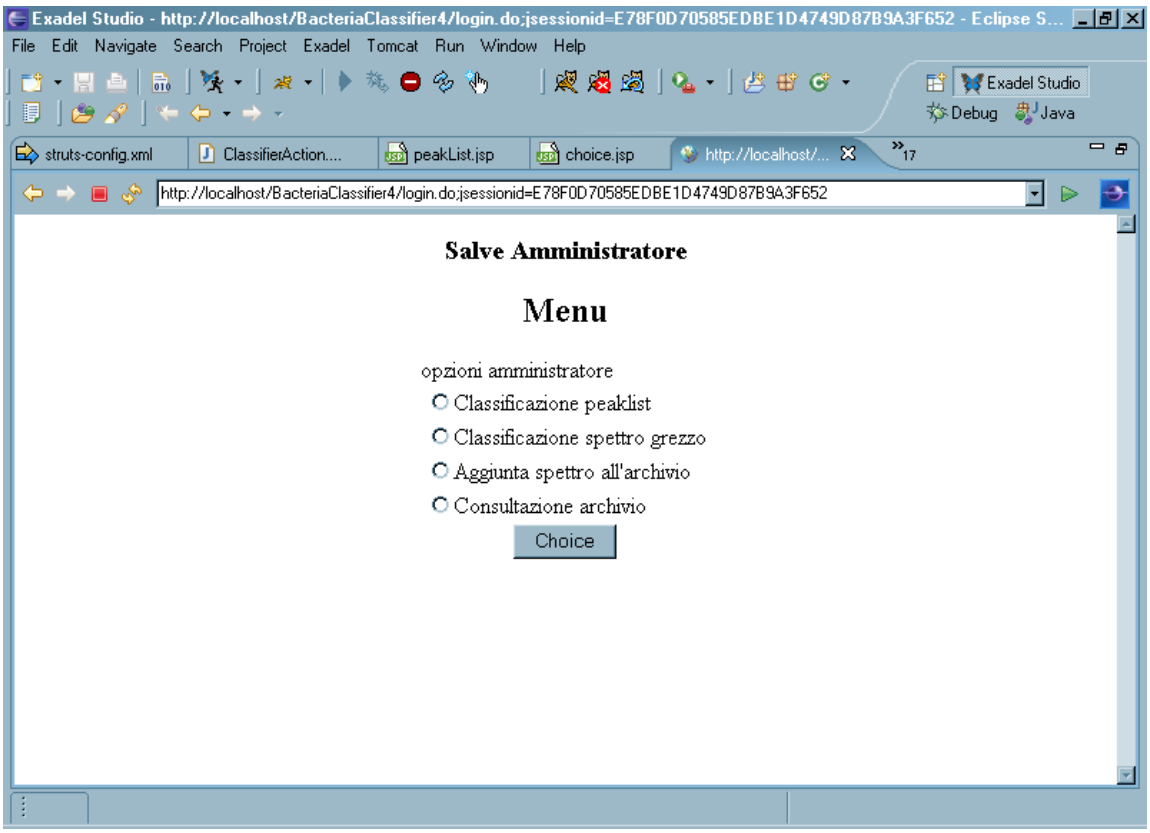
Nel caso di configurazioni non valide dei campi, nella pagina del login appaiono dei messaggi di errore. La dinamica di questi messaggi di errore ricalca la tipica dinamica del passaggio di informazioni attraverso l'applicazione. In particolare, se il form bean della pagina di login (il 'loginform'), rileva violazioni con il metodo validate, scrive nella sessione degli ActionError (o ActionMessage, per le versioni più recenti di Struts) relativi all'errore, che vengono poi raccolti dalla pagina login.jsp corrispondente, che li usa per costruire una nuova vista utente in cui questi messaggi appaiono. In questa dinamica, prende parte anche un file di properties dell'applicazione in cui vengono registrate coppie chiave-stringa. Gli ActionMessage registrati nelle sessione contengono delle chiavi identificative, attraverso le quali le pagine jsp raccolgono, dal file properties, il messaggio relativo da visualizzare. Tale separazione tra le cause d'errore e i messaggi da presentare, permette di ottenere un certo disaccoppiamento tra la gestione degli errori e l'interfaccia utente. In realtà, attraverso il file o i file di properties, si separa dalla view tutto ciò che riguarda la presentazione di testo e immagini all'utente. Ciò è sfruttato tipicamente per realizzare l'internazionalizzazione dell'applicazione. Segue un tratto del file properties che riguarda i messaggi di errore:

```
#errors associated with the Login page
errors.header=<font color="red">*

error.username.required=username is required.
error.password.required=password is required.
error.login.invalid=Username or password not correct!.
```

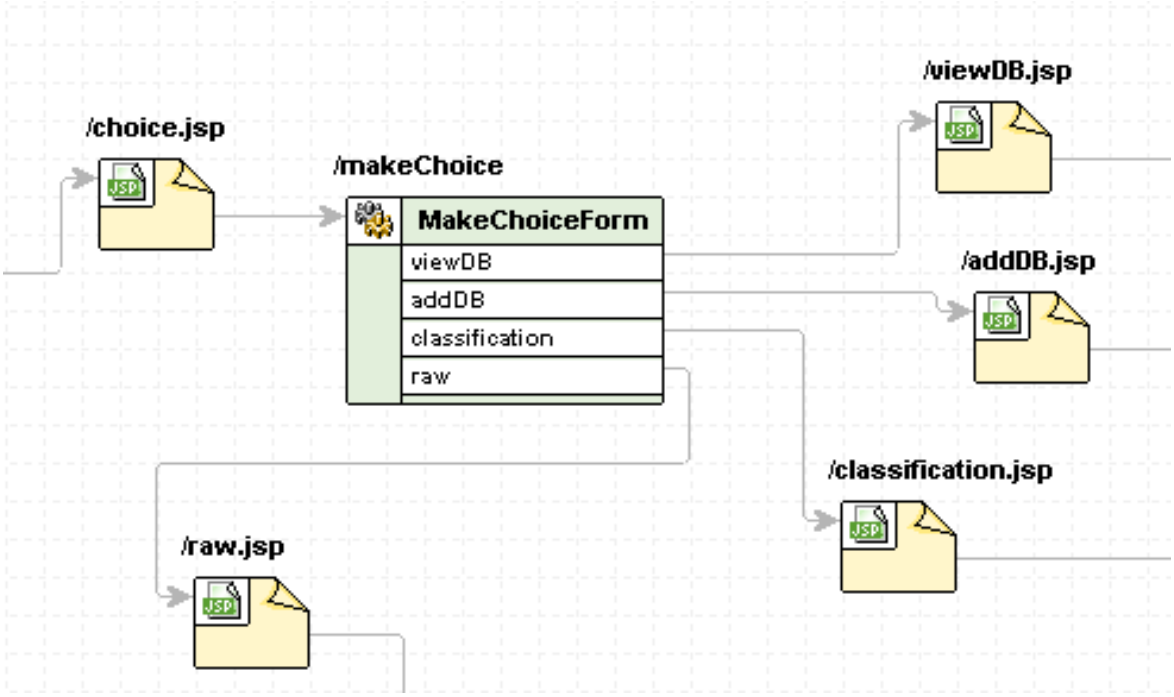
Attraverso le coppie chiave-messaggio e le tag, le pagine jsp possono visualizzare messaggi ed altre informazioni(vd. Par 3.27: Passaggio di informazioni.).

Par. 3.18: Il menu principale

Il menu principale presenta una scelta tra tutti i servizi dell'applicazione e la vista relativa è affidata alla pagina *choice.jsp*.

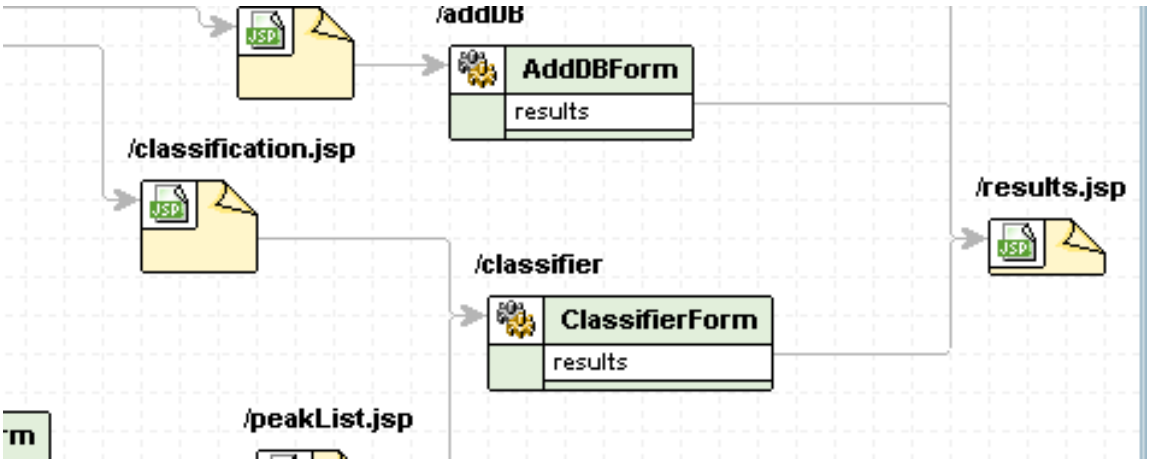


Nel caso di login accettato, tale prospetto si presenta all'utente. Ogni scelta è gestita dalla Servlet Action makeChoice . Questa action effettua un test per stabilire quale servizio, tra classificazione di spettri grezzi, classificazione di peaklist, archiviazione di uno spettro e ricerca spettro è stato scelto dall'utente. Questo snodo nel flusso di controllo poteva essere realizzato in diversi modi. Il modo più semplice è realizzare una Action per ogni scelta dell'utente. Ciò può avere dei vantaggi, come il distribuire il controllo di varie scelte su diverse Action, parallelizzando la gestione di diverse richieste. Ciò è utile in un contesto multithread, in cui una stessa istanza di una classe elabora varie richieste in concorrenza. Un altro modo per realizzare tale snodo è attraverso l'uso di una DispatchAction con la dinamica e i vantaggi visti nel paragrafo 3.15 . In questo caso si è scelto un metodo intermedio tra i due, in cui la Action, come visto, effettua un test su tutte le scelte possibili. Questa scelta è poco modulare, accorpando diverse dinamiche, rispetto alle Action separate, ma evita il proliferare di troppe Action nell'applicazione. La DispatchAction, oltre ad essere poco modulare, è complessa da manutenere, ma il problema principale è che porta più facilmente ad errori implementativi. La *makeChoiceAction* distribuisce il controllo sui vari rami dell'applicazione, attivando le relative viste.



Par. 3.19: La classificazione di una peaklist

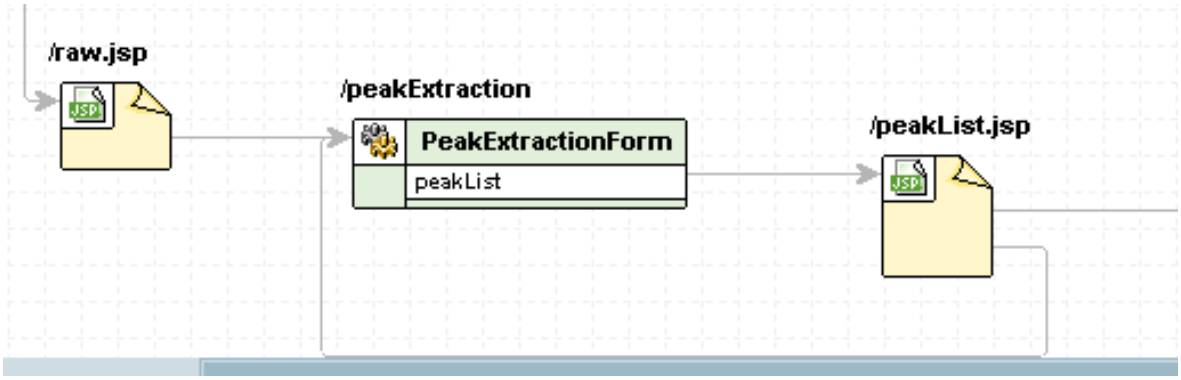
Una peaklist è uno spettro già elaborato da un esperto o automaticamente che presenta solo le informazioni utili alla sua identificazione. La peaklist può essere, quindi, trattata dallo stadio di classificazione, per il riconoscimento del batterio relativo. Quando un utente sceglie di classificare una peaklist, la *makeChoiceAction.java* avvia un Bean che accede al database per raccogliere i classificatori addestrati disponibili e quindi passa il controllo alla pagina *classification.jsp*, che costruisce una vista con i classificatori disponibili da scegliere e un campo per l'upload del file della peaklist. Una volta che l'utente ha scelto il classificatore e ha caricato il file della peaklist, con un certo formato, il controllo passa alla *classifierAction.java* che avvia, attraverso il *ClassiferBean.java*, il processo di classificazione.



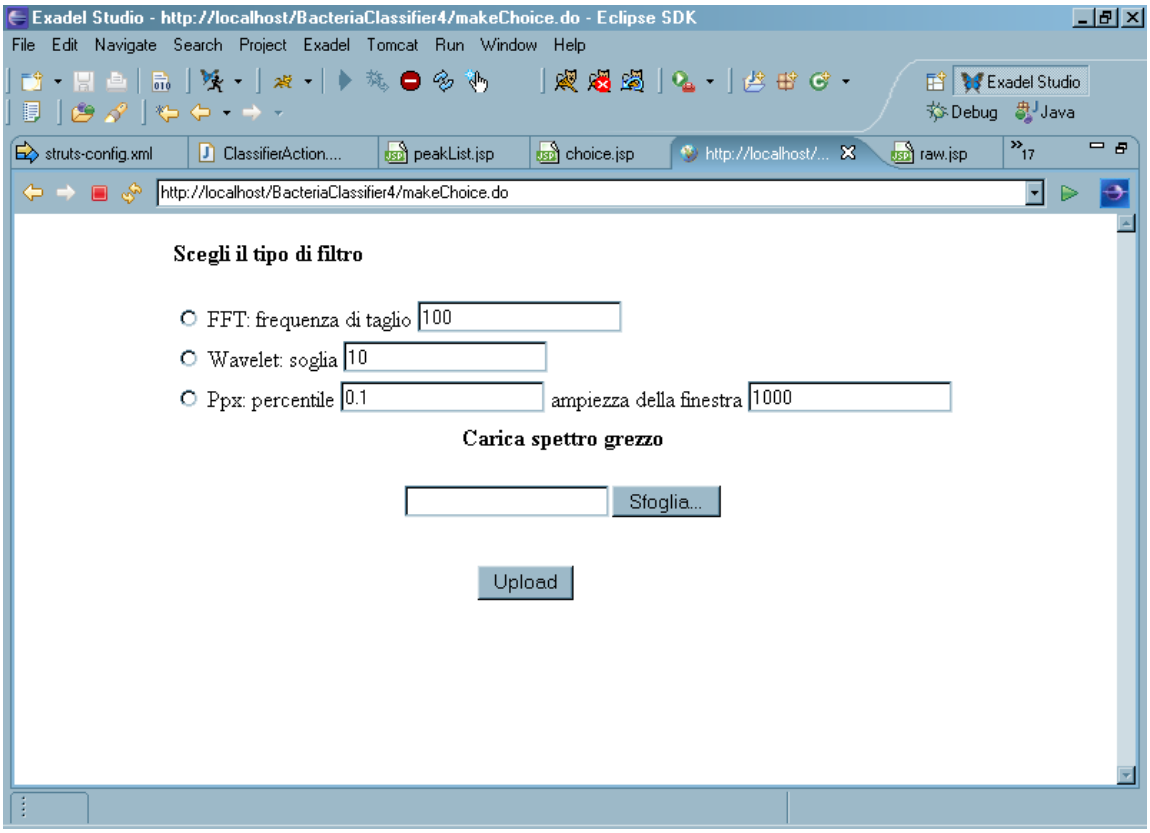
Il processo di classificazione, coinvolge le features extraction per la peaklist e la classificazione dello spettro. La classificazione dello spettro avviene leggendo dal database i dati binari del classificatore, convertendoli in oggetto e classificando mediante la dinamica e gli strumenti offerti dal WEKA. Il WEKA è un pacchetto open source scritto in Java, che raccoglie molti algoritmi riguardanti l'intelligenza artificiale, tra cui molti classificatori. Esso è stato sviluppato dell'Università di Waikato in Nuova Zelanda. I risultati ottenuti dal classificatore vengono poi presentati all'utente.

Par. 3.20: La classificazione di uno spettro in forma grezza

Uno spettro in forma grezza presenta, informazioni ridondanti e comunque non utili al suo riconoscimento. Innanzitutto, è corrotto da rumore sia chimico che elettrico ed inoltre non tutti i picchi presenti rappresentano in maniera univoca le caratteristiche del batterio: cioè non tutte i punti dello spettro possono essere considerati biomarkers. Tutte queste informazioni aggiuntive possono inficiare profondamente le prestazioni del classificatore, rendendo inutili le sue valutazioni. Quindi, quando uno spettro è grezzo va elaborato da alcuni algoritmi che ne eliminano il rumore, ne ricavano i biomarkers, costruiscono un relativo vettore delle features e lo classificano. Questa è la parte più articolata dell'applicazione.



La vista generata da raw.jsp presenta all'utente varie scelte possibili per estrarre da uno spettro grezzo la relativa peaklist. Segue una vista della pagina raw lato client.



Vi sono tre metodi per estrarre tale peaklist ed ognuno ha dei parametri da scegliere. In realtà, dietro ogni metodo di estrazione vi una metodo di pulizia dal rumore, a cui segue un relativo stadio di estrazione di picchi. Ad ogni metodo di filtraggio del rumore corrisponde un metodo di individuazione dei picchi. In particolare sussistono i seguenti accoppiamenti:

- filtraggio basato su FFT con individuazione dei picchi per confronto fra segnale filtrato e non;
- filtraggio mediante Wavelet con individuazione dei picchi per confronto fra segnale filtrato e non;
- filtraggio ed individuazione dei picchi mediante metodo Pepex-like.

Ogni metodo presenta dei parametri caratteristici, che influenzano la quantità di rumore rimosso e i picchi rilevati. Tali parametri sono lasciati alla discrezione dell'utente, poiché non esiste una scelta ottimale per ogni spettro. Inoltre si è deciso di lasciare la libertà all'utente di stimare, secondo la sua esperienza, cosa è utile e cosa, invece, può essere considerato rumore o superfluo in uno spettro. La dinamica implementata permette di verificare le scelte fatte da un utente, mediante la presentazione di una preview dello spettro elaborato: se l'utente non è soddisfatto ha la possibilità di ripetere l'elaborazione con parametri diversi. La realizzazione della preview della peaklist ha comportato alcune complicazioni nell'applicazione. Per mostrare la preview all'utente vi sono varie scelte:

1. utilizzando una applet,
2. utilizzando il display di un immagine sulla pagina utente generato a run-time,
3. salvando l'immagine in una directory temporanea.

La prima scelta è quella più ovvia, ma ha degli svantaggi. La applet deve essere caricata dall'utente comportando un certo carico sulla rete: se l'utente ha una connessione lenta, questo può non essere accettabile. Inoltre si deve considerare il principio di raggiungibilità: se l'utente, per motivi di sicurezza, ha disabilitato le applets, questa soluzione non funziona, limitando l'usabilità dell'applicazione. Il display di un immagine a run-time sulla pagina utente, comporta dei problemi di portabilità sul server ed è legato a certe funzionalità presenti su esso. In realtà, non si ha bisogno solo dell'immagine dell'elaborazione da presentare all'utente, ma anche di memorizzare lo spettro grezzo sottoposto dall'utente per le successive elaborazioni eventuali e le elaborazioni già effettuate per evitare di ripeterle in caso di riscontro positivo. Questa considerazione ha fatto propendere per la terza alternativa, che permette anche la gestione di risultati temporanei. La terza soluzione è più semplice da implementare, ma ha due gravi inconvenienti:

1. la gestione di una directory temporanea
2. il caching dal lato client.

Una directory temporanea sul server, con tutte le elaborazioni intermedie, può riempirsi facilmente e deve essere liberata dagli elementi non più necessari non appena possibile. Inoltre, la locazione di tale directory non deve essere fissa e legata alla particolare implementazione di sviluppo. La sua locazione deve

essere dinamicamente individuata a run-time, in modo da non compromettere il funzionamento dell'applicazione ad ogni migrazione della stessa.
Il caching lato client fa in modo che un immagine visualizzata dal browser sia memorizzata localmente. Questo è un vantaggio, perché le stesse immagini possono essere visualizzate più volte, ricorrendo alla copia locale, evitando di richiederle ogni volta dalla rete. Il browser controlla automaticamente che un immagine già visualizzata sia presente nella cache e quindi non la richiede al server. Nel caso delle preview delle elaborazioni, queste possono variare, ma il browser dell'utente visualizzerà sempre quelle memorizzate localmente. Forzare il browser utente a scaricare sempre nuove copie di un immagine ad ogni richiesta, anche se queste hanno lo stesso nome e url, è possibile.
Si deve imporre tale comportamento dalla parte server, ma ciò spesso non funziona, e se talvolta funziona non è detto che lo sia per ogni client. In particolare, tale variabilità è legata alla configurazione protocolli,sistema operativo, browser di ogni utente.

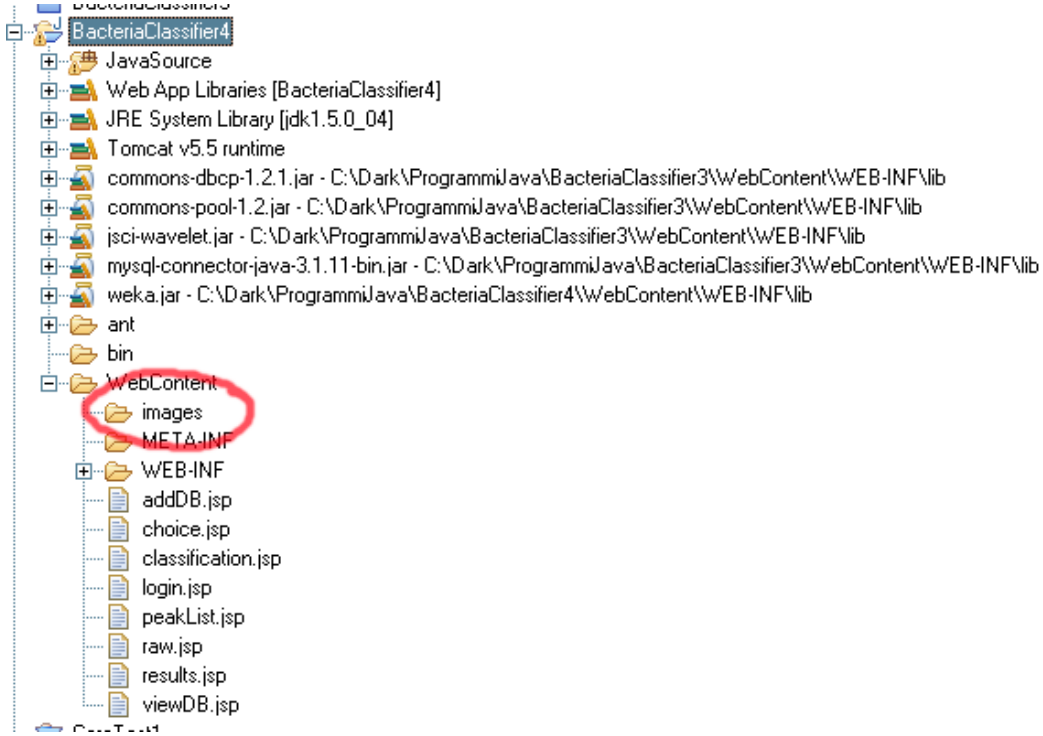
Par. 5.21: La gestione dei file temporanei

Lo svuotamento della directory dei file temporanei è stato delegato alla *LoginAction* . Essendo un aspetto proprio della parte Web e non della logica applicativa, la sua gestione è stata affidata alla parte *control* dell' MVC.
In sostanza, ad ogni login, la *LoginAction* controlla la directory dove sono allocati i file temporanei, ed elimina i file che sono stati creati da più di un giorno. Il tratto di codice che realizza tale funzione è il seguente:

```
//clear previous sessions images and temp files
final long sessionDur = 86400000; //one day

//get files of temp directory
ServletContext servletContext = servlet.getServletContext();
String imagesPath = servletContext.getRealPath("/images/");
File imagesDir = new File(imagesPath);
String[] fileList = imagesDir.list();



//delete old temp files
for( int i=0; i < fileList.length; i++ )
{
File tempFile = new File(imagesPath + "\\" + fileList[i]);
```

```
if(  System.currentTimeMillis() - tempFile.lastModified() > sessionDur )
 {
  tempFile.delete();
 }
```

In rosso sono evidenziati i tratti notevoli del codice. Nella prima riga evidenziata, si stabilisce la costante della durata di una sessione. In questo punto, si ipotizza che i file temporanei siano inutili dopo un certo tempo, cioè dopo che la sessione di lavoro è conclusa. In particolare, si ipotizza che le elaborazioni temporanee siano inutili dopo 24 ore e cioè dopo 86400000 millisecondi. La scelta di rappresentare il tempo in millisecondi rende certi confronti fra i tempi più semplici. I tratti successivi evidenziati, localizzano la directory temporanea, chiamata images. La directory images fa parte dell'albero delle directory proprie dell'applicazione:



Il problema è che non è possibile, dalle Actions dell'applicazione, accedere alle directory con path relativo a run-time; deve essere noto il path assoluto e tale path dipende dal modo in cui è istallata l'applicazione sul server. Leggendo il contesto della Servlet, si può rilevare il path assoluto della directory images, e quindi vi si può accedere. Nei passi successivi, si individuano tutti i file della

directory temporanea e si cancellano se sono stati modificati da più di un giorno. Questo confronto temporale, come accennato, è effettuato utilizzando i millisecondi, nell'ultima riga in rosso. Questa elaborazione rappresenta un certo sovraccarico di gestione per la Action che elabora il login.

Par. 3.22: Il caching lato client

Per evitare che il browser visualizzi sempre la stessa copia di un immagine, anche quando questa viene rielaborata dal server, si deve fare in modo che il suo nome cambi ad ogni nuova elaborazione.
La *PeakExtractionAction*,che scrive il file immagine della preview dell' elaborazione, decide il nome del file. Mentre il calcolo della peaklist relativa allo spettro grezzo e il disegno del relativo grafico sono delegeti a specifice classi della parte Model, la scrittura dell'immagine e il suo nome sono compito della *PeakExtractionAction*. Questa Action sceglie il nome del file con due accorgimenti:

- il nome del file deve essere legato alla sessione di lavoro,
- il nome del file deve cambiare ad ogni nuova elaborazione.

La Action rileva l'id della sessione e costruisce il nome del file con questo id. A tale nome concatena uno zero. Verifica che tale file esiste nella directory temporanea: se non esiste allora l'immagine viene scritta con il nome scelto, altrimenti si sostituisce lo zero finale con un uno e si cancella il file già presente.

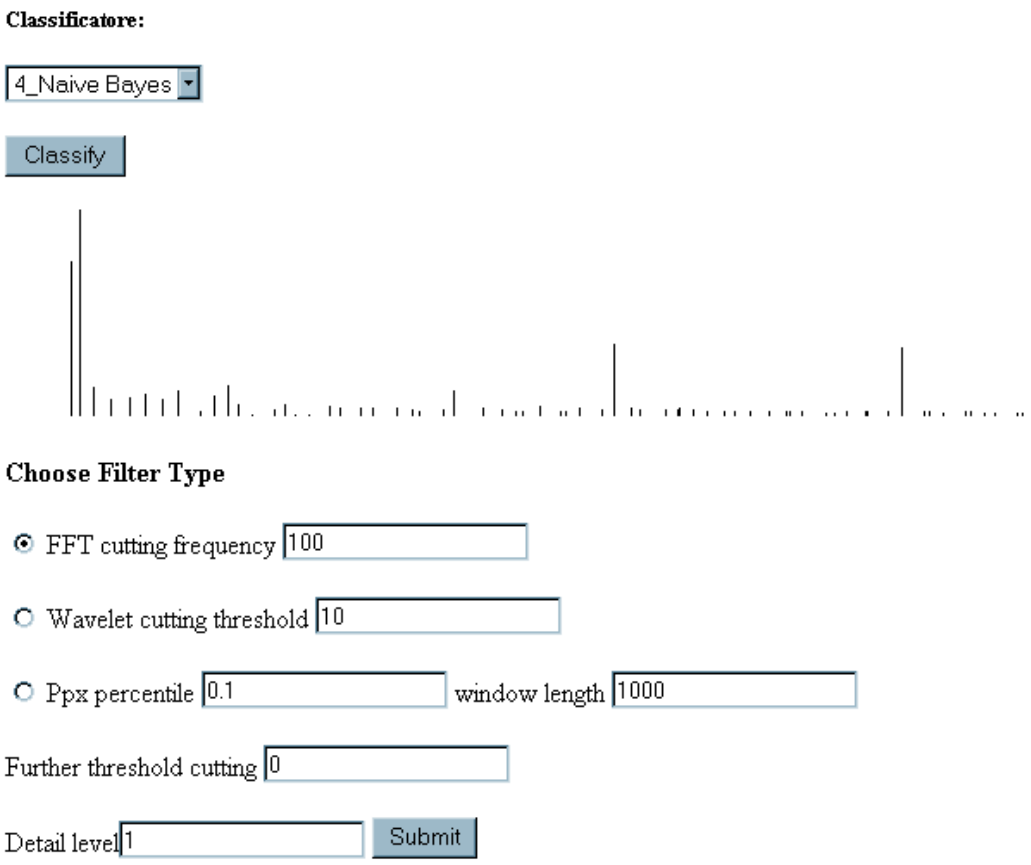
```
//choose file name for graph
ServletContext servletContext = servlet.getServletContext();
String imagesPath = servletContext.getRealPath("/images/");
String fileName=imagesPath + "\\" + session.getId() +"0.bmp";
File imageFile = new File(fileName);
if(imageFile.exists())
{
imageFile.delete();
fileName=imagesPath + "\\" + session.getId() +"1.bmp";
}
```

In questo modo i nomi dei file di elaborazioni successive presentano alternativamente zero ed uno alla fine, in modo da aggirare il sistema di caching del browser.
Il nome del file viene quindi passato alla pagina jsp relativa, legandolo come attributo alla sessione:

session.setAttribute("peakListFile", fileNamePL);

Par. 3.23: La preview della elaborazione dello spettro grezzo

In definitiva, la preview viene realizzata costruendo un immagine della peaklist elaborata e tale immagine viene posta in una directory temporanea a cui accede la pagina *peaklist.jsp* . Tale pagina costruisce una vista per l'utente in cui è riproposta la scelta dei parametri per una ulteriore elaborazione che viene gestita nuovamente attraverso *PeakExtractionAction*.
Per evitare il caching dal lato utente, il nome del file immagine viene cambiato in modo alternante, mentre la pulizia della directory images e delegata alla *LoginAction*. Nella figura seguente è mostrata la pagina peaklist spedita all'utente nella quale si nota il risultato di una estrazione dei picchi, di cui vi è la preview ed in cui vengono riproposte le scelte per eventuali ulteriori elaborazioni:



Par. 3.24: La classificazione dopo l'elaborazione dello spetro grezzo

Dopo l'estrazione della peaklist, tale spettro va classificato ed il controllo converge verso il ramo di classificazione già visto.
Realizzato il meccanismo di gestione della directory temporanea, si è scelto di utilizzarlo per memorizzare elaborazioni intermedie di spettri in modo da evitare, in caso di conferma, di ripetere elaborazioni onerose. Quindi la peaklist accettata viene letta dalla directory temporanea e passata alla *ClassificationAction*.

Par. 3.25: L' aggiunta di uno spettro al database e la visualizzazione di uno spettro presente.

L'aggiunta di uno spettro al database o la consultazione dello stesso hanno caratteristiche comuni per quanto riguarda l'implementazione con Struts. Tali caratteristiche, in realtà, sono presenti anche in altre parti dell'applicazione. Per la visualizzazione o l'aggiunta di uno spettro nel database, devono essere recuperate delle informazioni dal database perché l'utente scelga tra varie alternative. La visualizzazione degli spettri prevede una previa esplorazione del database per la ricerca degli spettri disponibili ed una visualizzazione di una scelta tra questi. L'aggiunta di uno spettro passa attraverso la visualizzazione dei nomi di batteri accettati dalla nomenclatura standard, tra cui l'utente deve scegliere. Tali dinamica preparatoria alle due funzioni è realizzata dalla *makeChoiceAction*, che raccoglie le informazioni attraverso accessi al database realizzati da opportuni java bean, e le rende disponibili alle pagine jsp scrivendo nella sessione opportuni attributi. L'accesso al database e il passaggio di informazioni attraverso la view, il model e il control sono realizzati attraverso tecniche messe a disposizione da Struts.

Par. 3.26: Accessi al database

Ogni accesso al database è realizzato da un java bean apposito, che la Action di turno invocano, per la raccolta di informazioni dagli archivi. La dinamica del collegamento al database deve avere delle caratteristiche:

- essere efficiente, per il numero di richieste da gestire;
- le informazioni per l'accesso al database non devono essere cablate nei java bean, ma devono essere separate e centralizzate per una facile riconfigurazione in caso di migrazione dell'applicazione.

Per realizzare tali caratteristiche vi sono delle tecniche che coinvolgono Struts ed altri elementi relativi al database.
Il coinvolgimento degli Struts nella gestione delle connessioni al database è una forzatura del modello MVC. Ciò che riguarda il database apparterrebbe al Model dell'applicazione e non alla View o al Control. In realtà, vi sono altri meccanismi per realizzare le connessioni con il database che non coinvolgono Struts, in modo da conservare al modularità e l'indipendenza dei moduli applicativi. In questo caso si è scelto di utilizzare Struts anche per questo aspetto. Il meccanismo di connessione al database implementato è quello che coinvolge un pool di connessioni e una classe detta DataSource.
Il pool di connessioni è una tecnica che istanzia un certo numero di connessioni al database rendendole disponibili per le richieste. Questo evita di dovere aprire una connessione al database ad ogni richiesta, rendendo più efficiente il servizio; infatti, la creazione di una connessione è un processo che costa in termini di risorse e tempo. Legata a questo meccanismo vi è la 'sorgente di dati' , DataSource che fornisce queste connessioni condivise fra gli utenti. Struts fornisce un componente di gestione di queste DataSource: attraverso il file *struts-config.xml* si può configurare la DataSource utilizzata dall'applicazione. Il tratto di *struts-config.xml* che segue illustra la configurazione utilizzata durante lo sviluppo dell'applicazione:

```
<!-- ========== Data Source Definitions
================================== -->
 <data-sources>
  <data-source key="BacteriaClassifier"
type="org.apache.commons.dbcp.BasicDataSource">
   <set-property property="description" value="Bacteria Classifier
Database"/>
   <set-property property="driverClassName"
value="com.mysql.jdbc.Driver"/>
   <set-property property="username" value="root"/>
   <set-property property="password" value=""/>
   <set-property property="url"
value="jdbc:mysql://localhost/spectra_bacteriorum"/>
  </data-source>
 </data-sources>
```

Nell'applicazione Bacteria Classifier si è utilizzata la classe DataSource fornita da Jakarta Commons Database Connection, che a sua volta utilizza Jakarta Commons Pool. Con questa tecnica si è resa più efficiente l'interazione col

database e contemporaneamente si sono separati i dettagli di connessione del database dalla logica applicativa. L'unico problema è che questi dettagli risultano appartenere ad un elemento della View (lo *struts-config.xml*), mentre la configurazione comune delle connessioni al database avrebbe dovuto far parte del Model. In altri termini la logica applicativa risulta legata alla parte Web dell'applicazione, aumentando la coesione fra le due e diminuendo la modularità.

Par. 3.27: Il passaggio di informazioni

Il passaggio di informazioni attraverso l'applicazione utilizza un meccanismo proprio delle Java Server Page. Ricorrendo a delle tag e agli attributi di sessione si possono trasferire informazioni da una parte all'altra dell'applicazione.
Un tipico percorso logico che riguarda questa tecnica consiste in

1. accesso al database,
2. raccolta di dati in un oggetto DTO,
3. registrazione di questo oggetto nelle sessione,
4. utilizzo di questo oggetto da parte di una pagina JSP per costruire una vista con le informazioni di interesse.

Con l'accesso al database avviato da una Action e realizzato da un Java Bean si raccolgono dati che devono essere visualizzati all'utente. Tali dati vengono registrati in un oggetto DTO (Data Transfer Object), realizzato in modo da consentire la memorizzazione dei dati e la loro consultazione.
I DTO sono dei Bean, che implementano un design pattern per risolvere proprio dei problemi di trasferimento di informazioni. Infatti, vengono popolati dai dati, all'atto delle loro creazione, dal Bean che accede al database e tali dati sono solo leggibili attraverso metodi di get e non più modificabili.
La Action registra tale oggetto nella sessione legandolo ad un attributo: questo attributo è utilizzato dalla pagina JSP, a cui la Action ha passato il controllo, per raccogliere le informazioni ed eventualmente visualizzarle.
Questo ultimo passaggio è realizzato mediante l'uso, nelle pagine JSP, delle tag della Java Server Page Tag Library (JSTL). In particolare, si sono utilizzate molto le tag della Core Library, ad esempio la tag <c:out per mostrare informazioni all'utente:

```
<c:out value='${spectra.type}' />
```

Questa tag è utlilizzata, ad esempio, per mostrare all'utente il tipo di spettro che si sta trattando.

La tag <c:forEach permette invece di iterare attraverso una collezione di oggetti DTO che rappresentano varie istanze di una stessa informazione. Con <c:forEach si possono raccogliere, in successione, tutti i dati degli oggetti DTO legati in un oggetto Collection, come nel tratto seguente:

```
<html:select property="classifier">
<c:forEach items='${classifierslist}' var='classifier' varStatus='status'>
<option><c:out value='${classifier.name}' />
</c:forEach>
</html:select>
```

In questo tratto di codice della pagina *peakList.jsp*, si utilizza il meccanismo per visualizzare i classificatori disponibili presenti in archivio.
Si notano inoltre anche le tag HTML, la cui libreria è fornita da Struts, che servono a rendere la scrittura di pagine JSP più semplice, con delle funzionalità non accessibili con le tag HTML standard.
Le tag HTML sono state usate ad esempio nella pagina *login.jsp*, per visualizzare i messaggi di errore sul login come visto nel paragrafo **5.17** . La potenza espressiva dei tag HTML di Struts è illustrata nel paragrafo seguente.

Par. 3.28: Upload di file

In alcuni punti dell'applicazione l'utente può fornire un file da elaborare. Per realizzare l'upload del file si è utilizzato il tag HTML file:

```
<html:file property="theFile"/>
```

Attraverso tale tag la pagina JSP realizza un campo per il Form in cui l'utente può scegliere dalle sue directory un file, che verrà spedito al server.. Il file viene passato ad un ActionForm che presenta una variabile di tipo FormFile la cui classe viene fornita da Struts:

```
import org.apache.struts.upload.FormFile;
```

Il file, così trasferito è disponibile alle successive elaborazioni o caricamenti nel database. Quindi, con Struts, il passaggio di un file da utente a server viene semplificato, almeno per quanto riguarda la sua codifica nell'applicazione.

Capitolo 4

BACTERIA CLASSIFIER: PARTE APPLICATIVA

Par. 4.1: Il dominio applicativo

Lo scopo dell'applicazione Bacteria Classifier è, approssimativamente, di raccogliere spettri batterici ottenuti con il Maldi-Tof , di immagazzinarli in un database e di cercare di classificarli, se la loro identità non è nota a priori.
La classificazione è la caratteristica più importante e critica dell' applicazione. Gli spettri di massa dei batteri hanno alcune caratteristiche particolari che rendono problematica una loro classificazione accurata o quanto meno accettabile.
Gli spettri di massa sono costituiti da molti punti, dell'ordine delle migliaia, e di questi punti alcuni contengono informazioni utili alla discriminazione tra spettri, mentre altri rappresentano il rumore dovuto a diverse cause (rumore elettrico dello strumento MALDI-TOF o masse che non appartengono al campione). Inoltre i campioni disponibili per ogni ceppo di batterio sono pochi ed è quindi difficile per un classificatore estrarre informazioni utili per la loro discriminazione. Queste due peculiarità producono effetti deleteri sulle prestazioni della classificazione.

Par. 4.2: La dimensionalità degli spettri

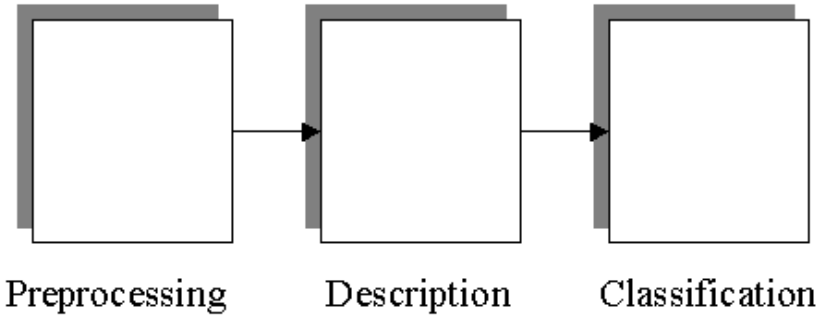
Ogni spettro di batterio è costituito da migliaia di punti, che rappresentano un andamento delle intensità delle masse in funzione dei rapporti carica–massa a cui è sovrapposto del rumore. Per la classificazione del batterio, questi punti vanno trattati e qualsiasi elaborazione su di essi può essere impegnativa: ogni punto è rappresentato da due valori reali, di cui uno è il rapporto massa-carica di uno ione, mentre l'altro è l'intensità del segnale relativo a questo rapporto. A questi valori si sovrappone del rumore dovuto a vari fenomeni tra cui il rumore elettrico dello strumento e l'influenza di eventuali masse non appartenenti al batterio. Questo rumore deve essere attenuato per non inficiare la capacità di un classificatore di discernere tra diversi spettri di batteri. L'elaborazione di questo rumore e la dimensionalità dello spettro rendono questo compito non banale.

Par. 4.3: La disponibilità dei campioni

Ogni campione da classificare presenta delle caratteristiche peculiari che ne identificano l'appartenenza ad una specifica classe. Questo è sfruttato da operatori umani per discriminare, in una collezione di dati, diverse categorie di appartenenza dei campioni. In uno spettro di massa batterico, le caratteristiche che permettono la classificazione sono rappresentate dai picchi di intensità di massa più elevati detti biomarkers. Nell'ambito della classificazione artificiale si cerca di rendere automatica l'individuazione di queste caratteristiche peculiari. In ogni caso il numero di questi biomarkers rilevati è dell'ordine delle centinaia e questo rappresenta un problema. E' infatti noto, da esperienze nell'ambito dell'intelligenza artificiale, che il numero di campioni disponibili per una classe deve essere dalla cinque alle dieci volte superiore al numero di caratteristiche significative. Questo rapporto è detto 'sample per feature ratio' (SFR) e influenza le prestazioni di un classificatore. Con un SFR basso si può avere un'accuratezza del classificatore scarsa ed un suo comportamento contraddittorio. Con gli spettri di batteri si incorre in questa anomalia, infatti, il numero di biomarkers è superiore al numero di campioni disponibili. In altri termini, gli spettri di massa per ogni tipo di batterio sono relativamente pochi rispetto alle caratteristiche significative necessarie per individuarli.

Par. 4.4: Il sistema di classificazione di Bacteria Classifier

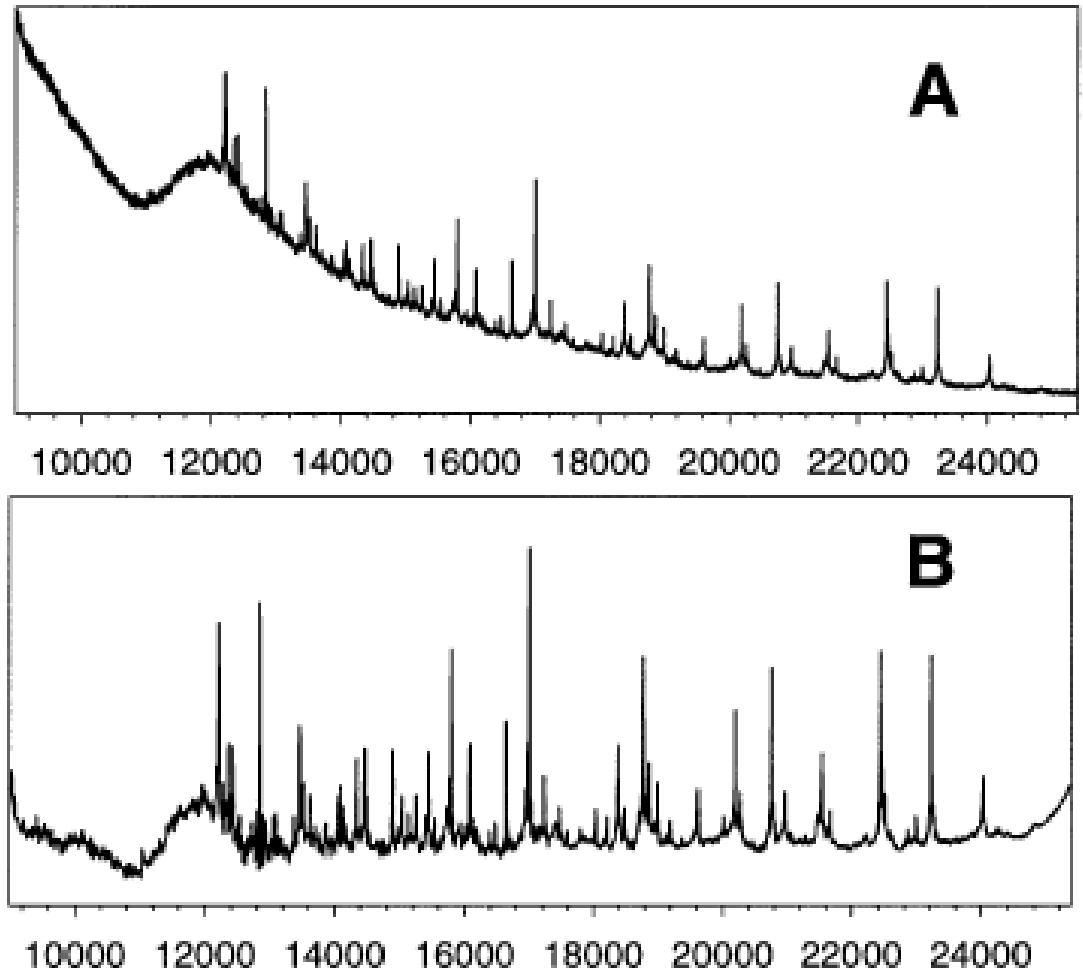
L'applicazione Bacteria Classifier, contiene quindi un sistema di classificazione degli spettri di massa batterici ottenuti con il MALDI-TOF.
Un sistema di classificazione è costituito da vari stadi, ognuno dei quali svolge vari compiti.



Approssimativamente, lo stadio di 'Preprocessing' elabora i dati in ingresso che nella maggior parte dei casi sono rumorosi o confusi da informazioni non utili. Lo stadio 'Description' estrae dai dati le caratteristiche importanti per la prototipazione dei campioni, in questo caso le caratteristiche sono i picchi significativi, i biomarkers. Lo stadio 'Classification' ha la responsabilità di discriminare tra i vari campioni sottoposti.

Par. 4.5: Il Preprocessing

Il preprocessing è il primo stadio che i dati incontrano nel loro percorso verso la classificazione. La dinamica della classificazione lavora su una gran mole di dati, che, nel caso della classificazione di batteri, è costituita da alcune informazioni importanti e da una gran quantità di informazioni irrilevanti. La gestione di questa massa di informazioni irrilevanti è un aspetto importante del riconoscimento dei batteri basato sulle teorie dell' Intelligenza Artificiale . Il problema è la selezione delle caratteristiche rilevanti dei dati rozzi da fornire al classificatore. Gli spettri rozzi di massa batterica possono essere di pessima qualità, con un rapporto segnale-rumore molto svantaggioso. Quindi è necessario elaborare i dati, prima di sottoporli allo stadio di classificazione e, in particolare, creare uno stadio di filtraggio, il preprocessing, che cerchi di separare le informazioni utili, da quelle irrilevanti. Infatti è noto che le prestazioni di un classificatore sono influenzate profondamente dai primi stadi di filtraggio e dalla loro capacità di eliminare il rumore. In altri termini, al classificatore devono arrivare le caratteristiche rilevanti dei dati perché esso sia in grado di rappresentarli ed il primo passo è la separazione del segnale dal rumore.
Questa separazione deve essere un processo rapido, robusto, e poco dipendente dall'operatore.
In uno spettro di massa batterico rozzo sono individuabili, tipicamente, una linea di base, 'baseline', e del rumore. La baseline è dovuta alla massa della matrice in cui è immerso il campione della coltura e che viene rilevata dallo strumento.Il rumore è dovuto al rumore elettrico dello strumento e da eventuali contaminazioni di elementi non appartenenti ai batteri della coltura.
Nelle ultime versioni del Maldi-Tof la baseline viene rimossa dallo strumento stesso ed inoltre, alcuni strumenti effettuano anche una smussatura dei picchi rilevati.



Lo spettro indicato con 'A' presenta le baseline, mentre lo spettro 'B' è il risultato di una prima elaborazione che il Maldi-Tof effettua sui segnali in uscita. Ciò che rimane è il rumore che deve essere efficacemente rimosso.
Sono stati seguiti vari metodi, per questa rimozione, ognuno dei quali si riferisce a diversi approcci teorici che derivano da altre discipline, come l'elaborazione digitale dei segnali, o il filtraggio di immagini. Lo spettro approssimativamente pulito dal rumore, va successivamente elaborato per la rilevazione dei picchi significativi, cioè vanno individuati i biomarkers. Tale operazione è generalmente delegata ad un esperto, ma vi sono alcuni algoritmi abbastanza efficaci per questo compito. Alcuni studi hanno confrontato i risultati di alcuni di questi algoritmi con le valutazioni di un gruppo di esperti, dimostrando, che dal punto di vista della classificazione automatica, non si hanno variazioni rilevanti delle prestazioni finali. Inoltre è dimostrato che gli insiemi di biomarkers, per ogni spettro, variano da un operatore esperto ad un altro e quindi non si può parlare di un insieme 'vero' di picchi identificativi di uno

spettro. In ogni caso l'operatore umano è molto efficace nell'individuare i picchi significativi la cui intensità è prossima al rumore; in altri termini, l'esperto è molto efficace sugli spettri deboli (molto corrotti dal rumore), mentre gli algoritmi sono più efficaci sugli spettri forti. In ogni caso gli operatori esperti lavorano con questi algoritmi di peak detection, regolando i loro parametri per una valutazione ottimale dei biomarkers. **[22]**
In definitiva lo stadio di prepocessing nell'applicazione Bacteria Classifier ha il compito di eliminare per quanto possibile il rumore dagli spettri di massa e di rilevarne i picchi significativi. Lo stadio di preprocessing, in effetti, cerca anche di ridurre la dimensionalità dello spettro, raccogliendo le informazioni più significative.

Par. 4.6: Il Preprocessing mediante FFT.

Uno spettro di massa è una collezione di dati che rappresentano gli isotopi presenti nel campione analizzato. Esso non è un segnale con andamento in funzione del tempo, ed anche ordinandolo in funzione dei rapporti carica-massa m/z, non si potrebbe a rigore parlare di andamento o paragonarlo ad una serie temporale. In ogni caso è possibile considerare uno spettro come un segnale, tenendo sempre ben presente la sua natura originale. Questo comporta la possibilità del'utilizzo di teorie, strumenti e risultati appartenenti ad altri campi di ricerca e tecnologie. In particolare, riconoscendo che nel preprocessing si cerca di eliminare per quanto possibile il rumore elettrico e chimico, si può tracciare un parallelismo tra il filtraggio di un segnale qualsiasi, che contiene informazioni e rumore, ed il filtraggio di uno spettro di massa. Con questo parallelismo, si possono sfruttare, con la dovuta prudenza teorica, i risultati ottenuti nell'ambito dell'elaborazione digitale dei segnali. In questo ambito è usato, con buon esito il filtraggio e l'analisi dei segnali con la trasformata FFT.
Il filtraggio mediante FFT si basa su un assunto secondo il quale il rumore che uno spettro di massa presenta è periodico ed è quindi teoricamente separabile dai picchi che rappresentano masse isotopiche, che hanno una diversa distribuzione in frequenza. In questo caso il tempo e rappresentato dalla scala dei valori m/z , detti Dalton, mentre la frequenza è identificata dall'inverso di questi valori.

Par. 4.7: La FFT.

Quando si visualizza un segnale, tipicamente si visualizza un andamento di una grandezza in funzione del tempo. Questa grandezza può essere una tensione, una temperatura, o altro. Analizzare un segnale nel dominio del tempo fornisce diverse informazioni, ma spesso è utile analizzarlo nel dominio della frequenza. In questo dominio sono spesso più evidenti delle caratteristiche del segnale, altrimenti implicite. In questo dominio si visualizza l'andamento della grandezza in funzione della frequenza: si evidenzia, cioè, l'energia del segnale alle varie frequenze. Per segnali complessi, questa descrizione permette molte valutazioni. Senza addentrarsi in dettagli teorici, si accenna che la teoria di Fourier, comprese la serie di Fourier e la trasformata di Fourier, legano l'andamento nel tempo di un segnale al suo andamento nella frequenza. La trasformata di Fourier è data da

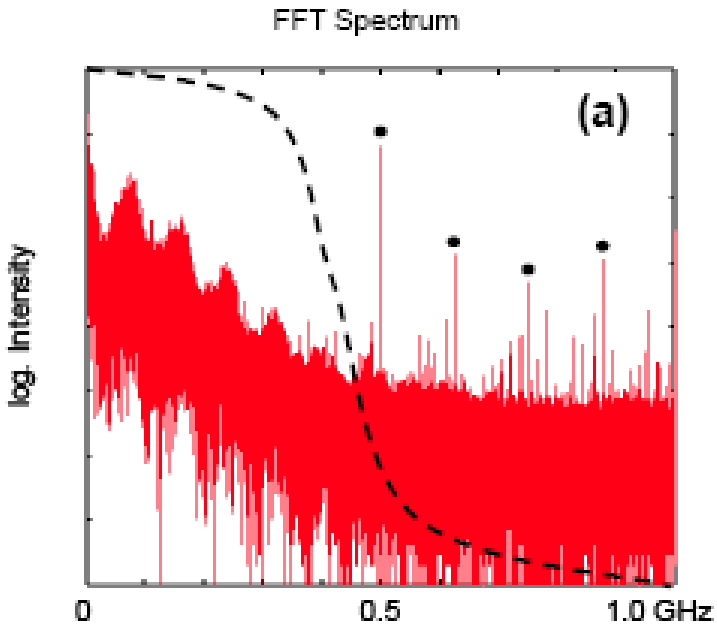
$$V(f) = \int_{-\infty}^{\infty} v(t)\, e^{-j2\pi ft}\, dt$$

La versione discreta della trasformata di Fourier è nota come DFT (Discrete Fourier Trasform).Questa trasformata elabora un segnale discreto nel dominio del tempo e ne computa la sua rappresentazione nel dominio della frequenza. La DFT è uno strumento utile per elaborare segnali del mondo reale. Per calcolare la DFT esiste un algoritmo molto efficace, che sfruttando alcune sue proprietà, genera lo spettro di un segnale, anzi genera una buona approssimazione della trasformata di Fourier del segnale.

Par 4.8: Il filtraggio mediante FFT.

[1] propone un metodo per il filtraggio di uno spettro di massa dal rumore e la rilevazione dei biomarkers, basato sulla trasformata FFT.
In pratica filtrando lo spettro e confrontando lo spettro filtrato con lo spettro non filtrato si possono rivelare i picchi significativi del campione.
Questo metodo si basa su delle assunzioni e valutazioni effettuate per gli spettri proteici, che sono poi state estese agli spettri batterici.

Secondo **[23]**, analizzando in frequenza uno spettro di massa proteico generato con il Maldi-Tof, si rileva una componente predominante in frequenza, di periodo 1 Da (Dalton). Questa componente, sempre secondo **[23]**, riflette la periodicità delle masse isotopiche presenti. Quindi, è toericamente possibile isolare il contenuto informativo di uno spettro di massa dal rumore, filtrando questa componente dalle altre presenti ad altre frequenze.



Nella figura sopra vi è uno spettro di massa proteico generato con il Maldi-Tof e trasformato con la FFT, che presenta rumore. Effettuando un filtraggio passa basso (linea tratteggiata) si ottiene un segnale che presenta una componente predominante, con periodo 1 Da, come in figura seguente.

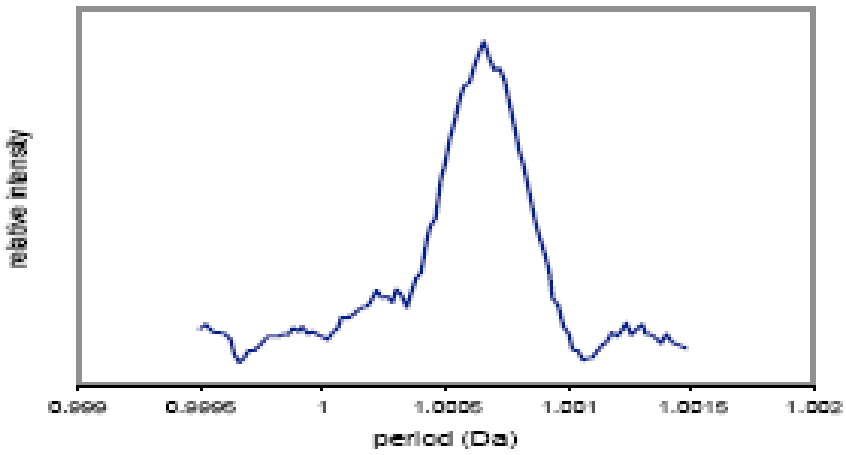
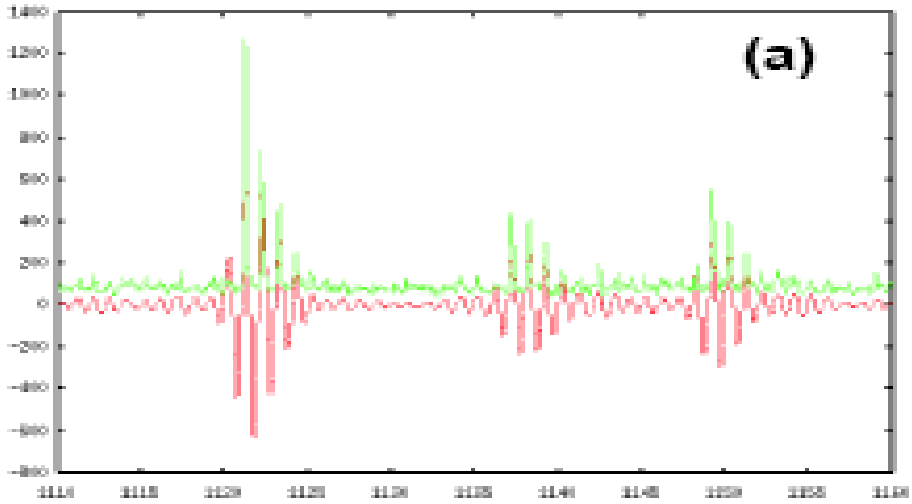
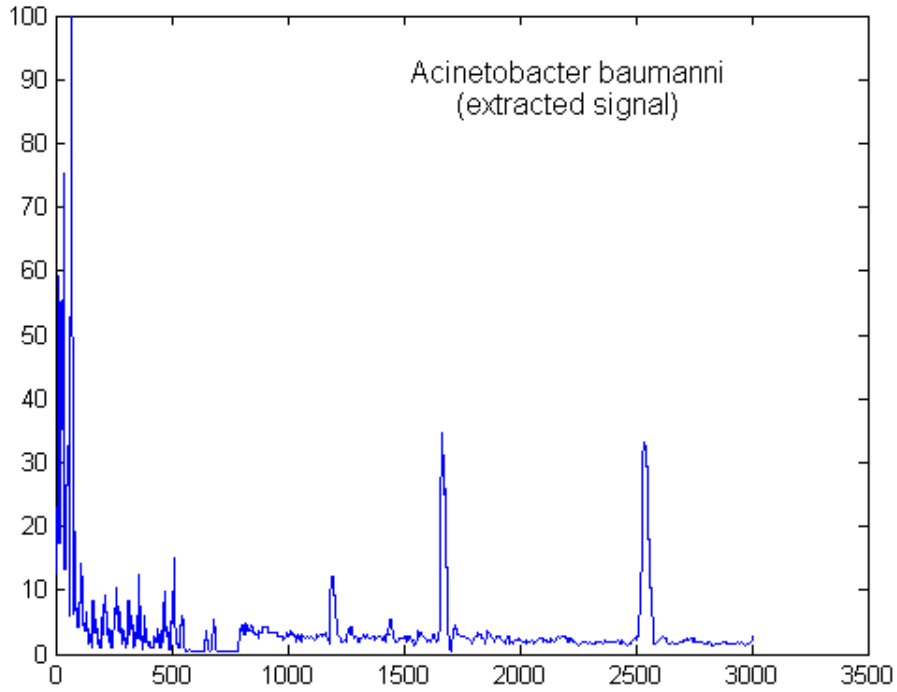


Figure 2. Frequency analysis of a MALDI-TOF peptide mass spectrum.

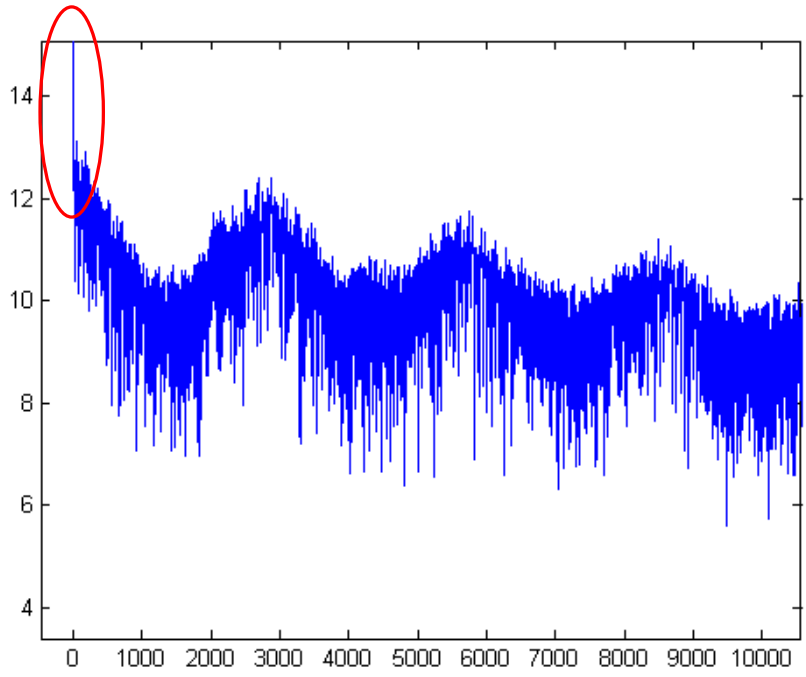
Filtrando quindi lo spettro e isolando le componenti a bassa frequenza si otiene un segnale le cui componenti rappresentano prevalentemente gli isotopi presenti nel campione, eliminando il rumore senza perdita di contenuto informativo.



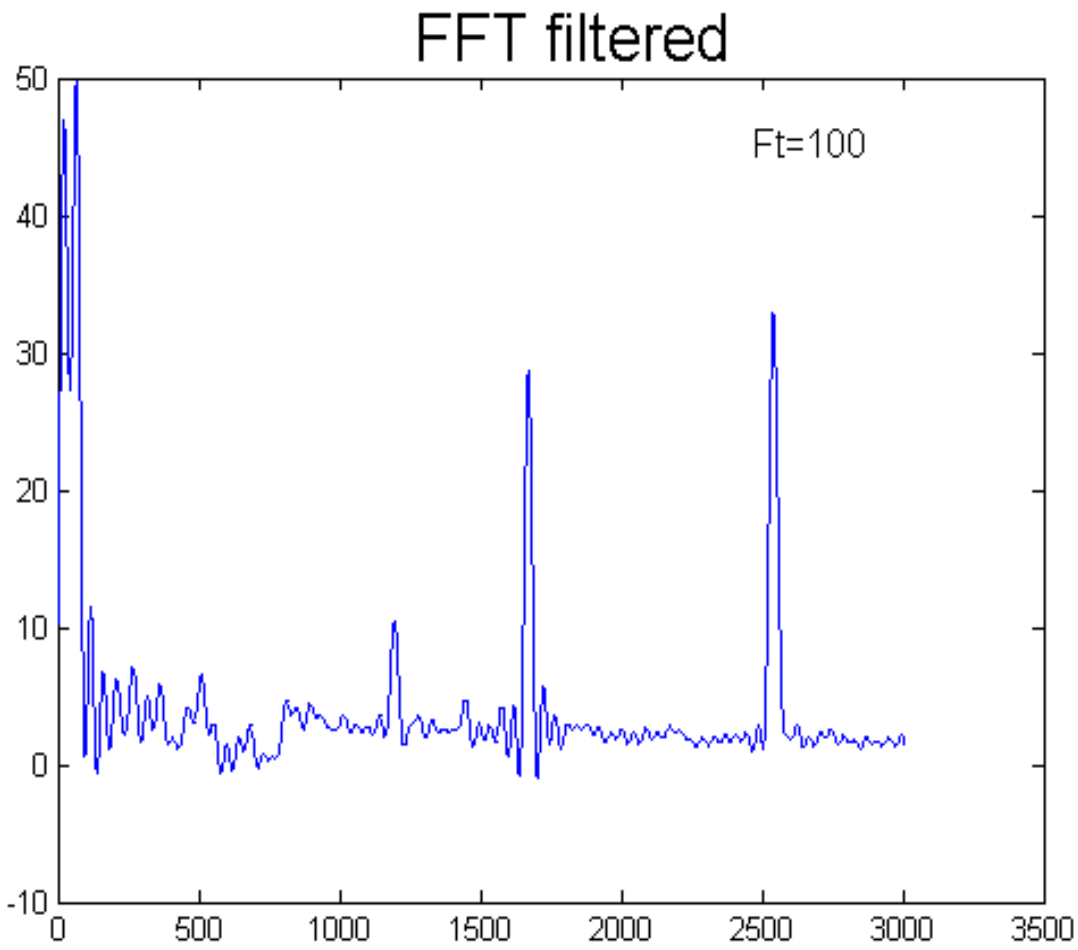
Nella figura sopra vi è il segnale grezzo,in verde e il segnale filtrato, in rosso, in cui si nota un andamento periodico, che secondo **[23]** è approssimativamente un'onda sinusoidale modulata dal segnale generato dagli isotopi presenti. Tutto ciò nell'ambito proteico. Per quanto riguarda gli spettri batterici si è seguito lo stesso percorso, cercando di verificare che le idee di **[23]** fossero applicabili anche a questi. Si è cominciato con lo spettro batterico di una coltura di Acinetobacter Baumanni:



Analizzando in frequenza lo spettro si è rilevata effettivamente una componente predominante a bassa frequenza(evidenziata in rosso nella figura seguente).



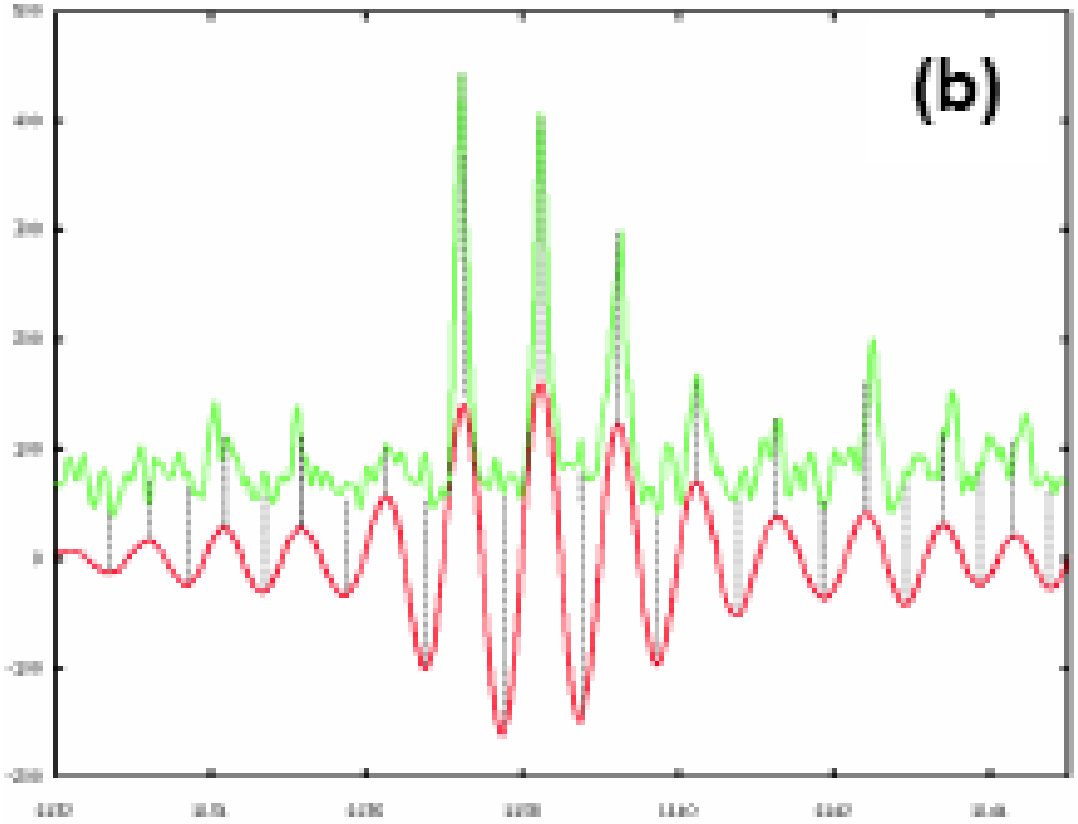
Filtrando lo spettro, eliminando le componenti ad alta frequenza, si è ottenuto un andamento simile a quello ottenuto da **[23]**. Il filtraggio è stato ottenuto con una frequenza di taglio abbastanza alta seguendo un approccio conservativo.



Par 4.9 La rilevazione dei picchi

Il metodo presentato in **[23]** prevede, dopo il filtraggio passa basso dello spettro, il confronto fra lo spettro filtrato e lo spettro originale. Lo spettro filtrato è utilizzato per individuare l'inizio e la fine di un picco, mentre lo spettro originale è utilizzato per rilevare l'intensità di questo picco.

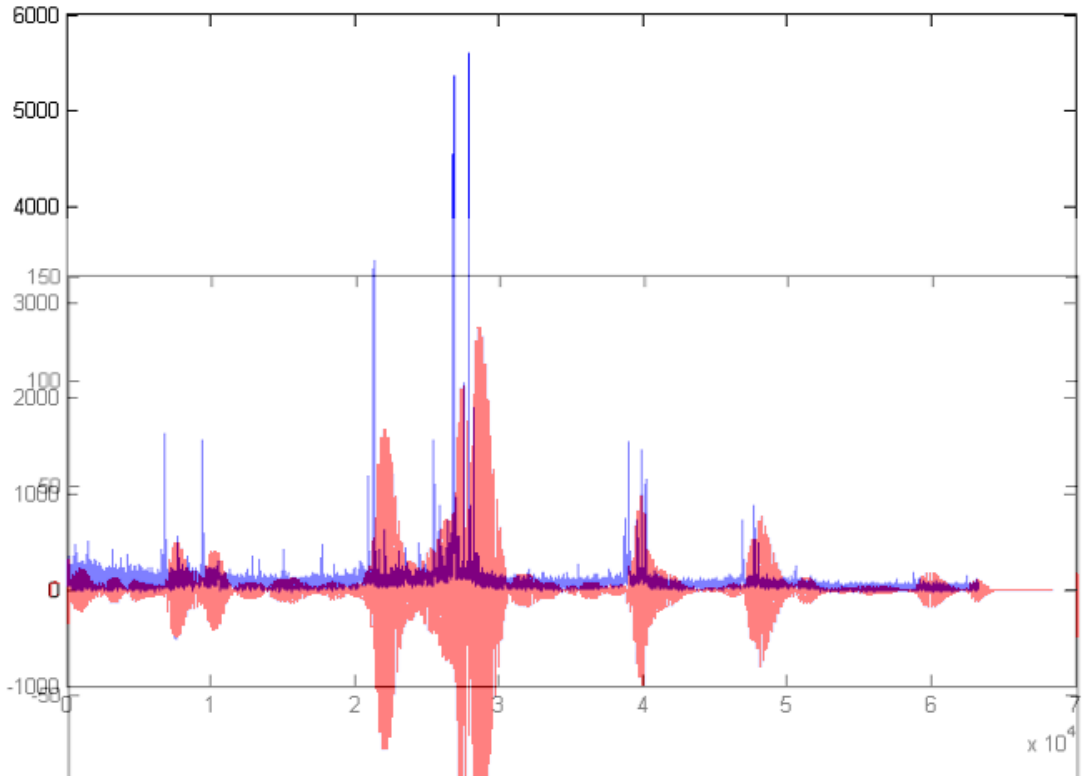
Lo spettro filtrato è un'onda con andamento periodico, con dei minimi e dei massimi. Si utilizano i minimi per localizzare un picco, infatti essi rappresentano la fine di un picco e l'inizio del picco successivo. Il massimo dell'onda tra questi due minimi serve ad individuare l'intensità del picco sul segnale originale. Il risultato di questa elaborazione produce una cosiddetta Peaklist (lista dei picchi) che rappresenta il contenuto informativo dello spettro, anche se con una certa approssimazione. L'obiettivo di questo metodo è l'eliminazione del rumore chimico ed elettrico, e l'eliminazione di informazioni ridondanti dallo spettro con l'effetto di una riduzione della dimensionalità dello stesso. La riduzione della dimensionalità dello spettro è sicuramente un aspetto desiderabile per i problemi che questa comporta nella classificazione.



Par 4.10 Implementazione del metodo

L'implementazione del metodo di preprocessing proposto da **[23]** non è stata banale. Nel metodo si susseguono due momenti: il filtraggio passa-basso e la peak detection. L'implementazione di un filtraggio passa-basso è stata difficoltosa; **[23]** non spiega come praticamente implementare il filtro ed elaborare il segnale.Lo spettro è rappresentato da un array bidimensionale con 2

colonne (valori m/z ed intensità delle masse) e diverse righe quanti sono i suoi punti. Si è scelto di elaborare questo spettro attraverso un filtro passa-basso in continua con il Matlab, ovviamente con risultati disastrosi. Si è scelto quindi di digitalizzare il filtro progettato in continua, utilizzando una certa frequenza di campionamento deducibile dalla spaziatura dei punti dello spettro e realizzando un filtro di Butterworth. A questo punto il segnale filtrato aveva perso le sue componenti ad alta frequenza, ma risultava sfasato rispetto all'originale. Come visto precedentemente, questo metodo impone l'allineamento dei segnali sui valori delle ascisse, cioè sui valori m/z, e lo sfasamento introdotto non è risolvibile in modo banale, nemmeno con una semplice traslazione del segnale filtrato. Scegliendo un filtro di Chebyshev, il fenomeno non si elimina. Nella figura seguente si sovrappongono il segnale grezzo, in blu, con il segnale filtrato, in rosso, ottenuto utilizzando un filtro a spillo: è evidente lo sfasamento



Il punto è che il Matlab simula un filtraggio per certi versi reale, che necessariamente introduce uno sfasamento sull'asse dei tempi, ma in questo caso non si può parlare, a rigore, di 'asse dei tempi'. Analizzando delle elaborazioni effettuate su onde sismiche si è capito che il filtraggio dei dati non

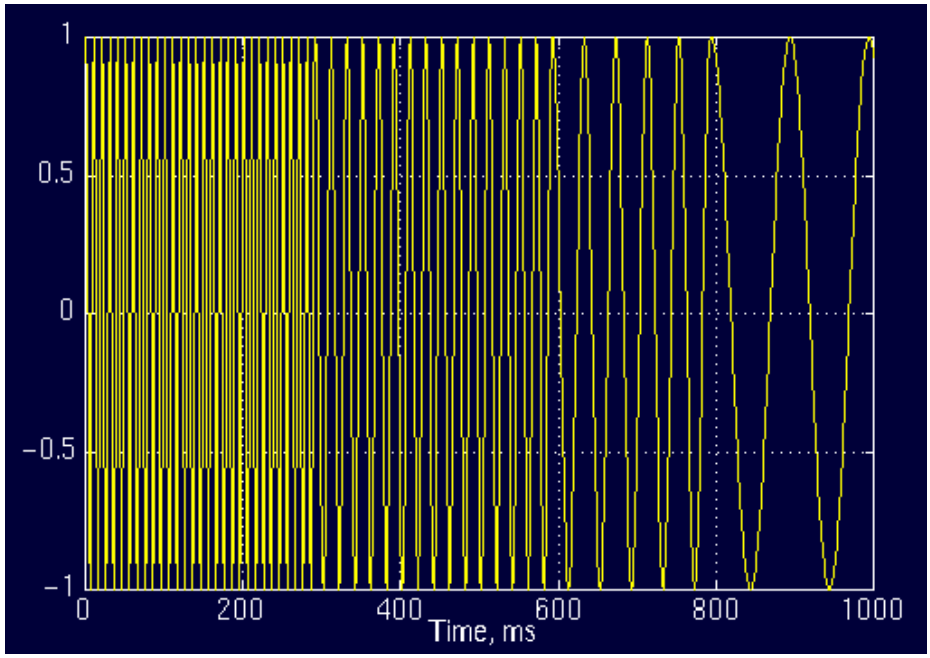
può tener conto di fenomeni fisici, quali lo sfasamento o distorsione. Il filtraggio degli spettri è in realtà un calcolo matematico e non è pertinente, in questo contesto, considerare tutta una serie di fenomeni che appartengono al filtraggio di segnali digitali. In particolare, l'analogia tra asse dei tempi ed asse delle m/z non è più pertinente. Quindi, con la prospettiva di un'elaborazione matamatica si è cercato di filtrare lo spettro senza introdurre altre dinamiche. Questo punto di vista permette di considerare quindi filtri ideali, realizzati attraverso la FFT. Si è trasformato lo spettro con la FFT e si sono eliminate le componenti ad alte frequenze che in una FFT rappresentano i punti centrali dello spettro. I punti centrali dello spettro sono individuati rispetto ad una frequenza di taglio scelta: i punti compresi nell'intervallo tra la frequenza di taglio e il limite dello spettro meno la frequenza di taglio rappresentano le componenti ad alta frequenza. Azzerando brutalmente questi punti si ottiene qualcosa di simile ad un filtraggio ideale dello spettro, che ha inoltre la proprietà di non essere traslato sull'asse delle ascisse. Anche se questa elaborazione non è ortodossa e non è dimostrata la sua pertinenza con la trasformata di Fourier dello spettro originale, si è ottenuto un risultato soddisfacente per l'implementazione del metodo **[23]** ed anche una certa rapidità di esecuzione. Il metodo, infatti, richiede in sostanza, che il segnale elaborato presenti un'onda modulata utile alla localizzazione dei picchi. Applicando il metodo si sono rilevati i biomarkers con una buona efficienza e con la possibilità di variare i picchi rilevati variando la frequenza di taglio del filtro. Inoltre si è ottenuta una rappresentazione dello spettro più compatta e consistente.

Par 4.11 Preprocessing mediante trasformata Wavelet

Il preprocessing basato sulla trasformata Wavelet comporta l'utilizzo di questa trasformata per l'eliminazione del rumore ed un meccanismo simile al metodo basato su FFT per la rilevazione dei picchi. L'utilizzo delle Wavelet per l'eliminazione del rumore da dati grezzi ha prodotto risultati soddisfacenti in molti campi, dall'elaborazione di immagini, satelitari e non, all'elaborazione di dati finanziari o onde sismiche. La trasformata Wavelet ha delle caratteristiche molto utili per l'elaborazione di spettri di massa. In particolare Coombes et al. Hanno investigato sull'uso di Wavelet nel preprocessing per alleggerire gli effetti deleteri del rumore su dati biologici nella loro classificazione. In particolare Coombes ha dimostrato che, se la località degli attributi di questi dati è abbastanza pronunciata, il filtraggio con Wavelet incrementa le prestazioni finali. Quindi, l'analisi Wavelet permette l'attenuazione del rumore, con certe caratteristiche, ed una efficiente compressione dei dati.

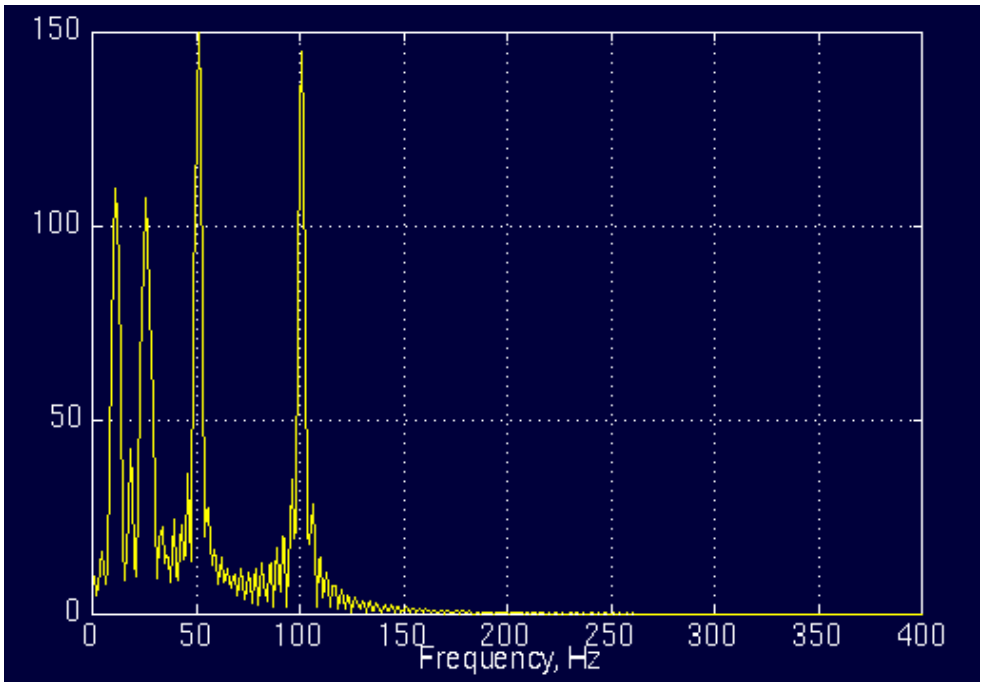
Par 4.12 La trasformata Wavelet

La trasformata Wavelet per certi versi ricorda la trasformata di Fourier. Non si entra nel dettaglio della teoria matematica che tratta questa trasformata, ma comunque si cerca di delinearne le caratteristiche principali. Come per la trasformata di Fourier, la trasformata Wavelet elabora i segnali nel dominio del tempo, cercandone una descrizione alternativa che ne metta in risalto alcune caratteristiche, cambiando il punto di vista con cui si analizzano questi segnali. Vi sono delle differenze notevoli tra le due trasformate. La trasformata di Fourier permette di passare dal dominio del tempo al dominio della frequenza, agevolmente, ma nel dominio del tempo non sono evidenti informazioni sulla frequenza, mentre nel dominio della frequenza non sono visibili informazioni sul tempo: con la trasformata di Fourier informazioni sul tempo e sulla frequenza di un segnale non sono mai contemporaneamente presenti. In pratica, la trasformata di Fourier mette in risalto le componenti in frequenza del segnale, ma non indica il momento in cui queste componenti appaiono nel segnale. Questa informazione può essere fondamentale in segnali non 'stazionari'. I segnali stazionari sono segnali in cui il contenuto in frequenza non cambia nel tempo, tutte le loro frequenze sono presenti in ogni momento.



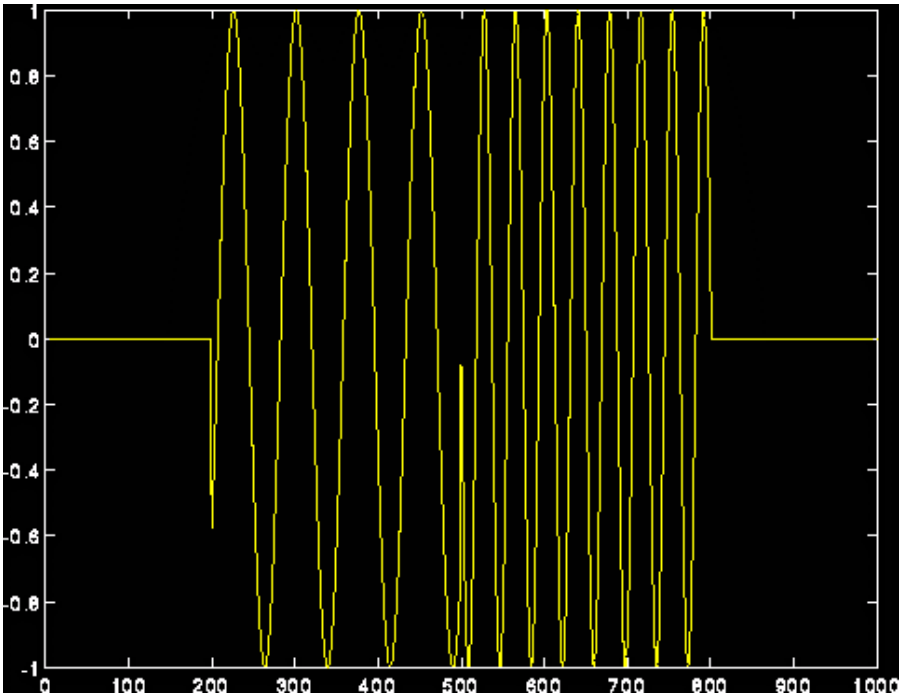
Nel segnale in figura le componenti in frequenza cambiano da un intervallo di tempo all'altro. Effettuandone la trasformata di Fourier si ottiene lo spettro

seguente in cui sono evidenziate tutte le componenti ma non è chiaro se queste componenti sono presenti sempre oppure in momenti definiti ed eventualmente in quali momenti:

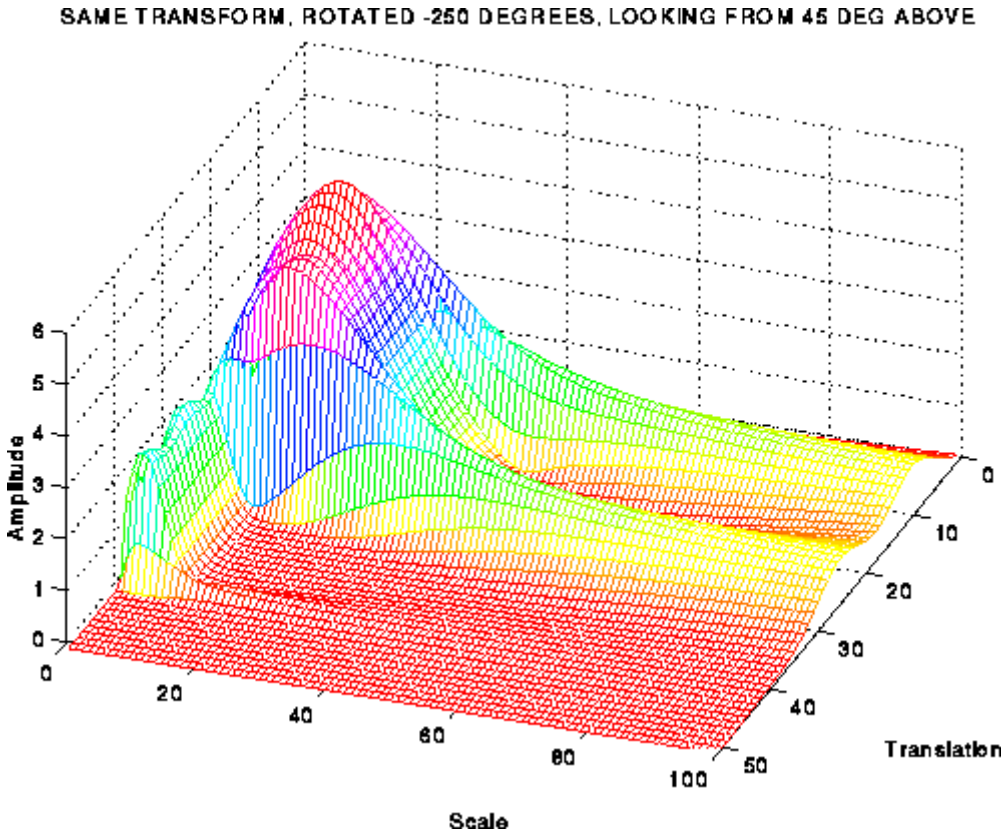


Lo spettro della figura sopra, in realtà, è molto simile allo spettro di un segnale in cui le componenti (quattro, rappresentate dai quattro picchi iniziali) sono sempre presenti e quindi la trasformata di Fourier non basta a rappresentare efficacemente segnali complessi. Facendo un'ardita generalizzazione si può affermare che quasi tutti i segnali di natura biologica sono di tipo non stazionario e quindi anche gli spettri di massa batterici. Per analizzare questi ultimi è necessario uno strumento più sofisticato della trasformata di Fourier. La trasformata Wavelet fornisce, appunto, una rappresentazione tempo-frequenza del segnale. In realtà, va menzionata un'altra trasformata, creata per risolvere questo problema, la STFT (short time Fourier Trasform). Questa trasformata, presenta un piccolo inconveniente da descrivere con una piccola digressione. Le informazioni sulla frequenza e sul tempo di un segnale in certo e ben definito punto nel piano tempo-frequenza non possono essere note. In altri termini non è possibile sapere quale componente spettrale è presente in un segnale in un certo istante. Questo è una sorta di principio di indeterminazione applicato ai segnali ed infatti, è possibile conoscere solo quali componenti spettrali sono presenti in un intervallo di tempo per un segnale. L'inconveniente della STFT è che essa fornisce una risoluzione costante , mentre la trasformata Wavelet

permette di variare la risoluzione del piano tempo-frequenza, cioè la dimensione degli intervalli in cui si analizza il segnale. Infatti è noto che le alte frequenze hanno una migliore risoluzione nel tempo , mentre le basse frequenze hanno una migliore risoluzione nella frequenza. In altri termini si può localizzare meglio una componente ad alta frequenza nel tempo rispetto ad una componente a bassa frequenza, mentre questa componente è meglio descritta nella frequenza. La trasformata Wavelet permette di variare la risoluzione, migliorando l'analisi del segnale.



Questo è un segnale di esempio, che verrà elaborato con la trasformata Wavelet e che presenta, all'inizio, una componente a bassa frequenza e verso la fine una componente ad alta frequenza.



In questa figura si nota una trasformata Wavelet: l'asse indicato con 'Scale' rappresenta l'inverso della frequenza. Si evidenziano due picchi: quello più ampio è relativo alla componente a bassa frequenza, quello più contenuto è relativo alla componente ad alta frequenza.
La trasformata Wavelet nel continuo è definita dalla seguente formula:

$$CWT_x^{\psi}(\tau, s) = \Psi_x^{\psi}(\tau, s) = \frac{1}{\sqrt{|s|}} \int x(t)\psi^* \left(\frac{t-\tau}{s}\right) dt$$

dove x(t) è il segnale. La trasformata è funzione di due variabili τ (traslazione) ed s (scala). La scala fornisce, per così dire, una variazione sui dettagli presentati dalla trasformata: piccoli valori della scala forniscono informazioni globali sul segnale, mentre alti valori forniscono dettagli crescenti circa il segnale. La funzione Ψ() è detta Wavelet (piccola onda) ed è una funzione che deve possedere delle proprietà per essere utilizzata cnella trasformazione In termini approssimati, la Ψ() deve essere di lunghezza finita e deve avere una natura, per così dire, oscillatoria. Matematicamente le funzioni Ψ() Wavelet devono essere

basi ortonormali di $L^2(\mathcal{R})$ ed in letteratura sono state proposte molte funzioni Wavelet. La trasformata Wavelet continua garantisce la perfetta ricostruibilità del segnale. Il punto è che si deve implementare un algoritmo su computer che valuti, o meglio, approssimi questa trasformata per un segnale. Un modo per calcolare praticamente la trasformata Wavelet è di discretizzare la trasformata, cioè di campionare la trasformata continua. Non entrando nei dettagli, la trasformata discretizzata può essere molto onerosa del punto di vista computazionale ed è da escludere per l'elaborazione di spettri di massa, costituiti da migliaia di punti. L'alternativa è l'utilizzo di della trasformata Wavelet discreta (DWT). La DWT fornisce sufficienti informazioni sia per l'analisi che per la sintesi di un segnale, con una riduzione del tempo di calcolo. La trasformata continua Wavelet è una correlazione tra una Wavelet a differenti valori della scala ed il segnale; tale trasformata è calcolata cambiando la scala della finestra, scorrendo la finestra nel tempo, moltiplicando per il segnale ed integrando. Trasferendo questa idea al caso discreto, si usano filtri a differenti frequenze di taglio per analizare il segnale a differenti scalature. In pratica, il segnale è passato attraverso una serie di filtri passa-alto per analizzarlo ad alta frequenza e filtri passa-basso per analizzarlo a bassa frequenza. Senza entrare in dettagli, la procedura inizia con il passaggio del segnale in un filtro digitale passa-basso con risposta impulsiva h[n] , cioè calcolando la seguente convoluzione:

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Il filtro passa-basso rimuove le frequenze che sono sotto la metà della banda del segnale. La frequenza per un segnale discreto è espressa in radianti e se il segnale deriva da un campionamento alla frequenza di Nyquist, la sua massima frequenza è π radianti. Dopo aver filtrato il segnale attraverso un filtro passa-basso, si possono eliminare da esso metà dei campioni, poiché ora il segnale ha una frequenza massima di $\pi/2$ radianti invece di π. Quindi si può sottocampionare il segnale filtrato e conservere solo metà dei suoi punti, infatti per decrivere il segnale filtrato passa-basso sono necessari solo metà dei campioni, dimezzando la risoluzione. La procedura può essere espressa come

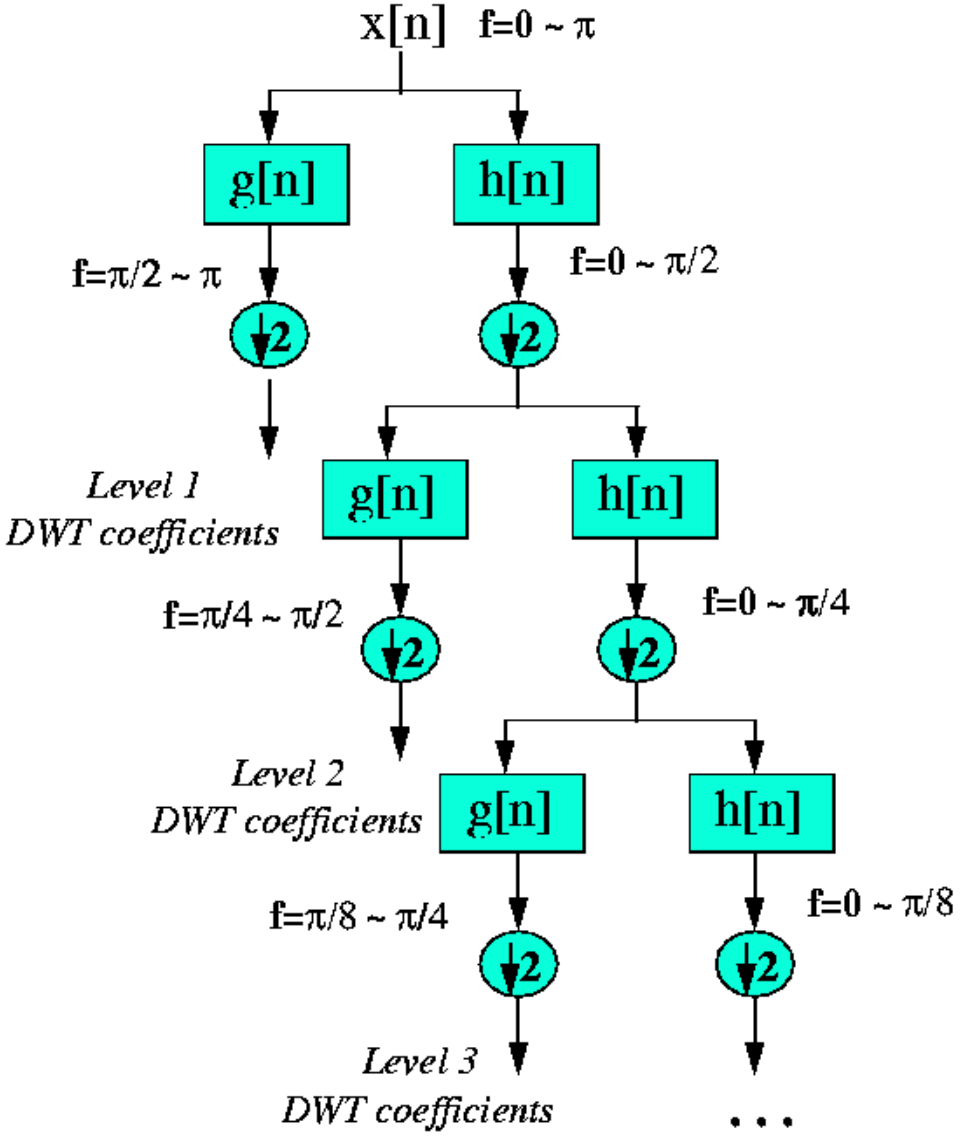
$$y[n] = \sum_{k=-\infty}^{\infty} h[k] \cdot x[2n-k]$$

Premesso ciò, è possibile delineare come la DWT viene calcolata: la DWT analizza il segnale a differenti bande di frequenza con diversa risoluzione, decomponendo il segnale in una parte ad approssimazione rozza ed una con maggiori dettagli. La DWT utilizza due insiemi di funzioni, le funzioni di scalatura (scaling) e le funzioni Wavelet che sono associate rispettivamente con i filtri passa-basso e passa-alto. La decomposizione del segnale in diverse bande di frequenza è ottenuta con successivi passaggi attraverso filtri passa-basso e passa-alto del segnale. Dapprima il segnale viene elaborato da un filtro passa-alto g[n] ed un filtro passa-basso h[n] e dopo questo filtraggio metà dei campioni del segnale viene scartato sottocampionandolo. Questa è la decomposizione di primo livello, descritta da :

$$y_{high}[k] = \sum_n x[n] \cdot g[2k - n]$$

$$y_{low}[k] = \sum_n x[n] \cdot h[2k - n]$$

dove $y_{high}[k]$ e $y_{low}[k]$ sono le uscite dei filtri passa-alto passa-basso rispettivamente. Questa decomposizione dimezza la risoluzione nel tempo (sottocampionamento), ma raddoppia la risoluzione in frequenza, poiché dimezzando la banda riduce l'incertezza in frequenza di metà. Tale procedura viene ripetuta per ulteriori decomposizioni. Ad ogni livello, il filtraggio ed il sottocampionamento riduce di metà il numero di campioni (dimezzando la risoluzione nel tempo) e di metà la banda (raddoppiando, invece, la risoluzione in frequenza). La figura seguente descrive il procedimento; x[n] è il segnale da analizzare, h[n] è il filtro passa-basso, g[n] è il filtro passa-alto, mentre la banda del segnale ad ogni livello è indicata con 'f':



Ogni livello di decomposizione restituisce una insieme di coefficienti e la DWT finale è ottenuta concatenando questi coefficienti cominciando da quelli dell'ultima livello. I coefficienti della DWT sono in numero pari ai punti del segnale di partenza. Le componenti spettrali predominanti del segnale appariranno con maggiore ampiezza nella zonea della DWT che riguarda quelle frequenze. La differenza con la trasformata di Fourier è che la localizzazione nel tempo di queste frequenze non viene persa. Tale localizzazione ha, comunque, una risoluzione che dipende da quale livello queste frequenze appaiono. Questa procedura offre una buona localizzazione temporale delle alte frequenze perché

sono rappresentate da molti campioni, mentre fornisce una localizzazione più scarsa per le basse frequenze che vengono descritte da pochi campioni. In ogni caso le componenti spettrali più deboli, appaiono con minore ampiezza e possono essere eliminate con un meccanismo a soglia che azzera i relativi coefficienti della DWT. In letteratura, sono state proposte molte funzioni di scaling (filtri passa-basso) e funzioni Wavelet (filtri passa-alto) per calcolare la DWT.

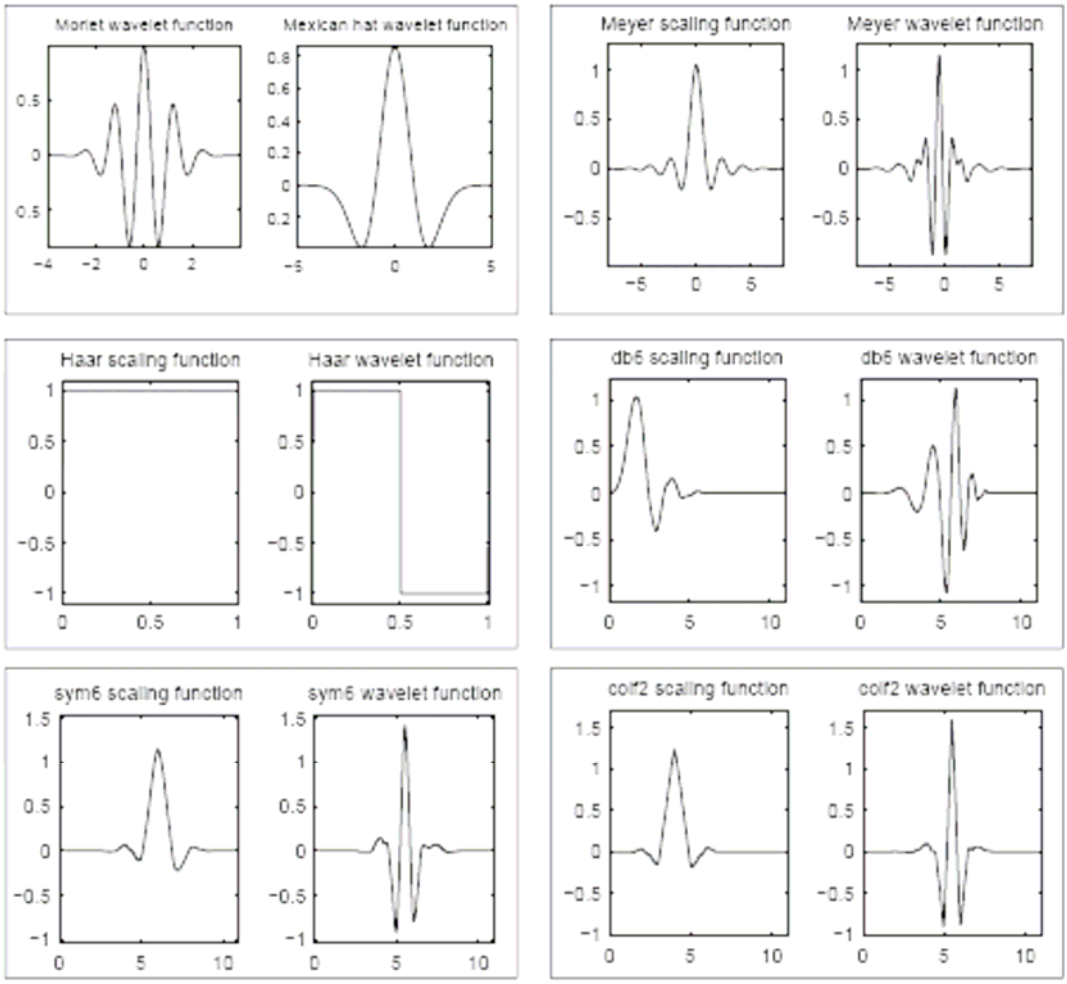
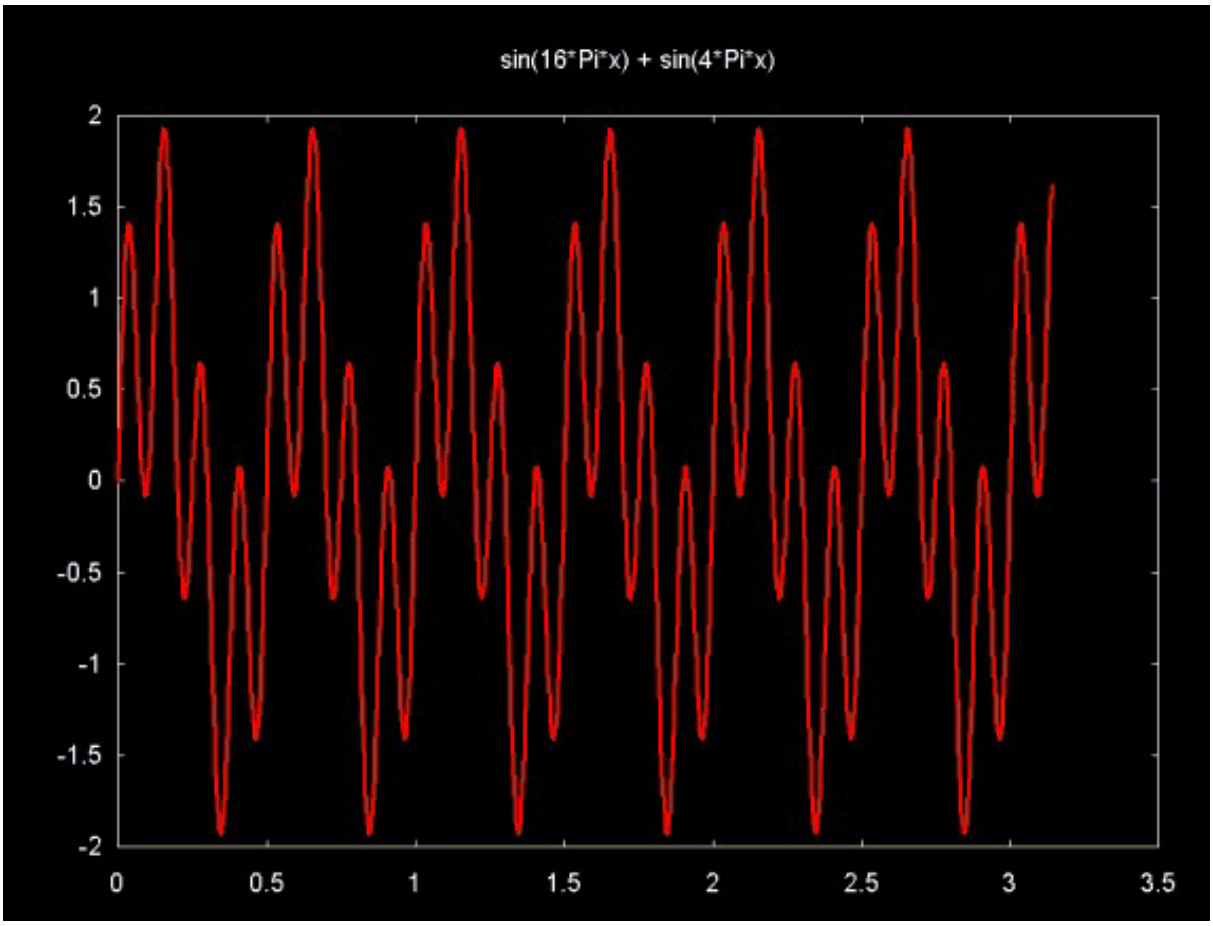


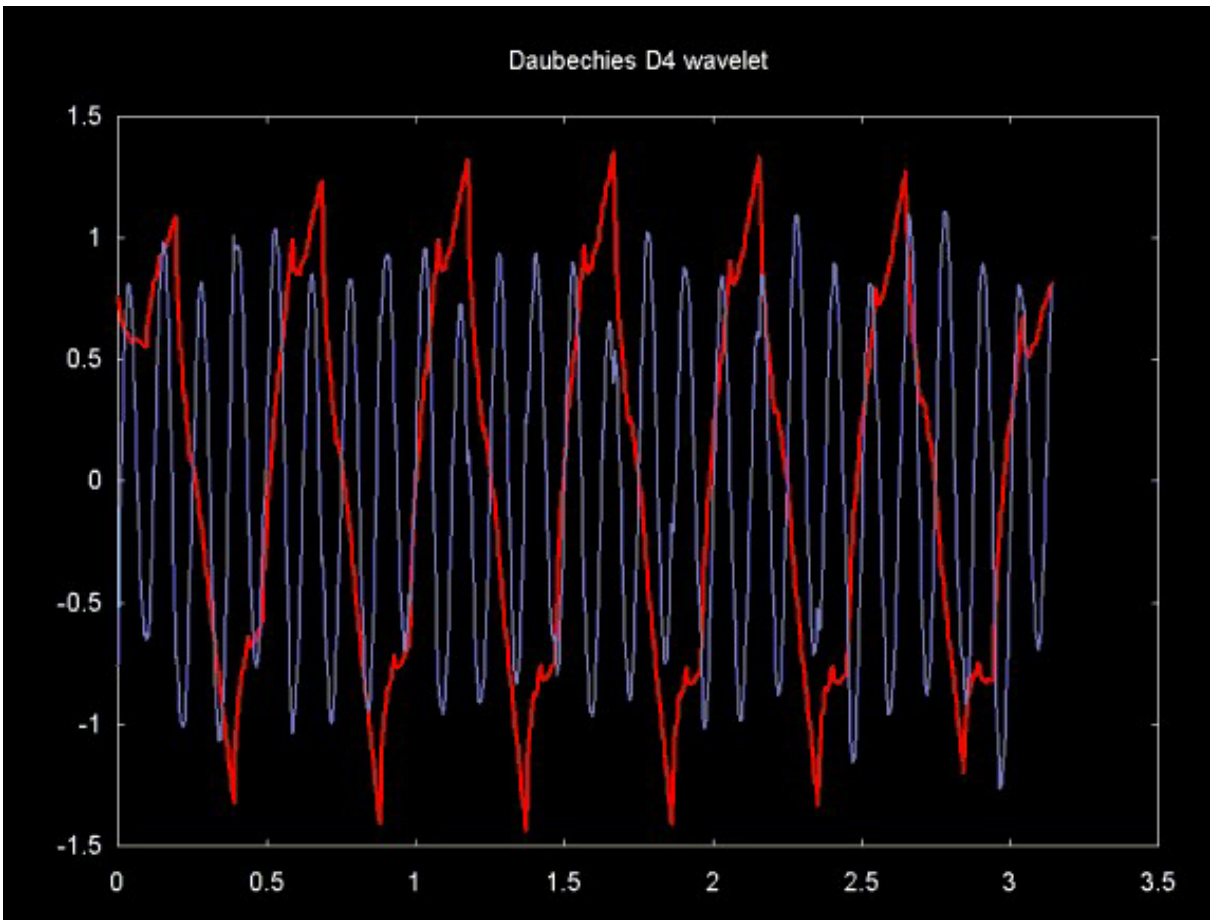
Figure 6-1 Various One-Dimensional Wavelets

Il punto è che la trasformata DWT non garantisce la perfetta ricostruibilità del segnale a causa del fatto che queste funzioni riescono solo ad approssimare i filtri ideali passa-basso e passa-alto. Quanto le funzioni proposte si avvicinano ai filtri ideali dipende dalla natura di queste funzioni e da come interagiscono col segnale di partenza. Ad esempio si analizza nel seguito come un segnale composto da due onde sinusoidali viene trattato dalla DWT. Si confrontano due

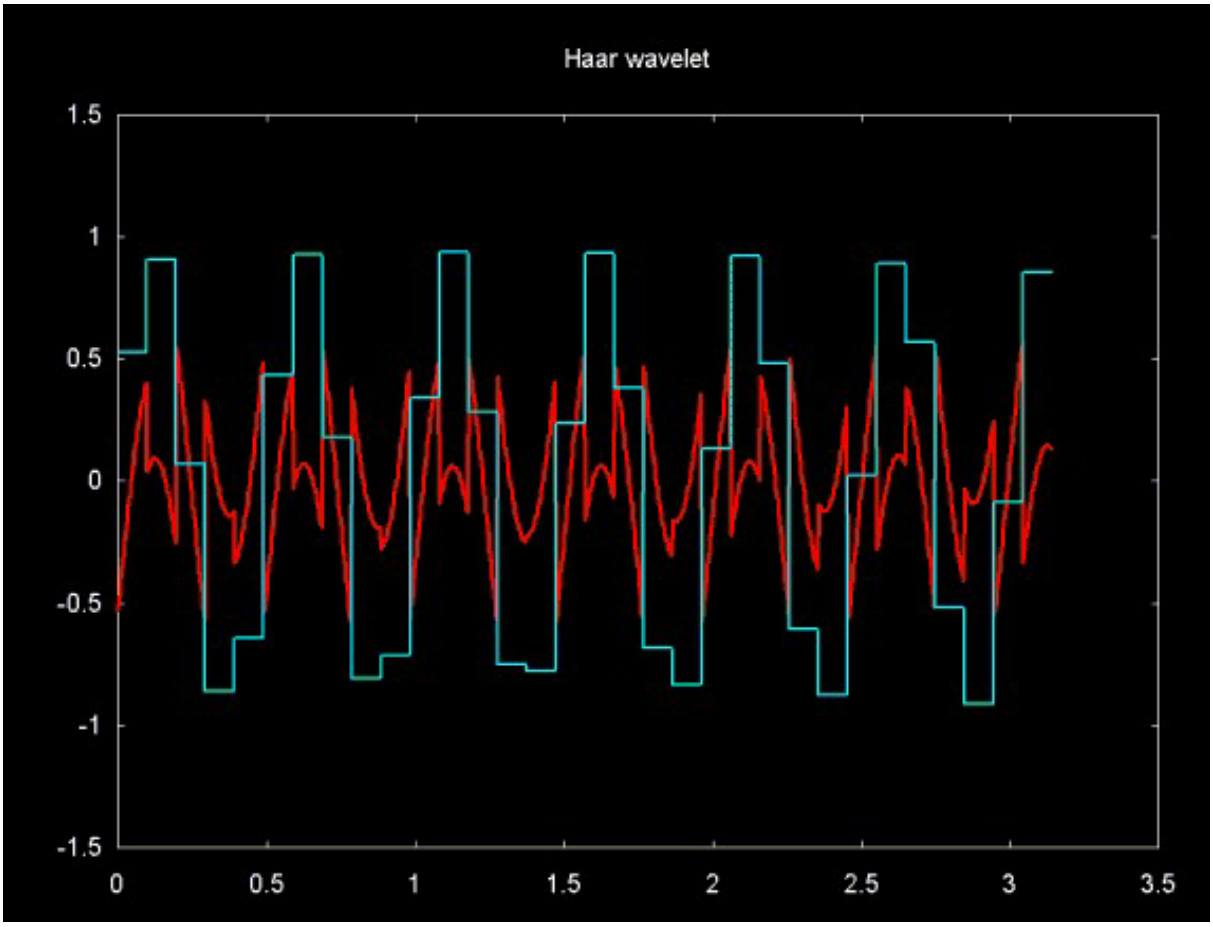
tipi di trasformate proposte dalla letteratura: le Wavelet di Haar e le Wavelet di Daubechies 4. In entrambi i casi si verifica la ricostruibilità del segnale dopo la trasformazione. Il segnale in ingresso è illustrato dalla seguente figura:



Ricostruendo il segnale dopo la trasformazione si ottiene per le Daubechies 4 il seguente grafico:



dove, in rosso, è tracciata la parte passa-basso, mentre, in blu, la parte passa-alto. La parte in bassa frequenza non segue l'andamento del segnale originale, ma l'andamento delle Wavelet Daubechies. Per le Haar la situazione è anche peggiore:



Realizzare un filtraggio di un segnale per attenuare il rumore utilizzando la trasformata Wavelet DWT è complicato da questa imperfetta ricostruibilità e dal fatto devono essere scelte con cura le funzioni della trasformata. Alcuni tipi di dati sono trattati meglio dalla trasforata di Haar, come i dati finanziari, mentre altri dati, come onde sismiche sono trattati meglio da altri tipi di trasformate, come le Daubechies.
Attenuare il rumore mediante trasformata Wavelet è in teoria possibile seguendo alcuni accorgimenti. In pratica si tratta di eliminare i coefficienti della trasformata che riguardano alcune frequenze. Un metodo per eliminare, ad esempio, il rumore Gaussiano da un segnale è proposto da Percival e Walden nel cap 10 del loro *Wavelet Methods for Time Series Analysis* **[15]**. In pratica questo algoritmo prevede di:

1. calcolare la trasformata Wavelet ed ordinare i coefficienti per frequenze crescenti

2. calcolare la deviazione media assoluta dei coefficienti data da

$$\delta(mad) = \frac{median\{|c_o|, |c_o|, \dots |c_{2^n-1}-1|\}}{0.6745}$$

(il fattore *0.6745* è utilizzato perché la δ_{mad} sia una stima della deviazione standard del rumore Gaussiano)

3. si calcola una soglia (formula proposta da D.L. Donoho e I.M. Johnstone)

$$\tau = \delta_{mad}\sqrt{\ln(N)}$$

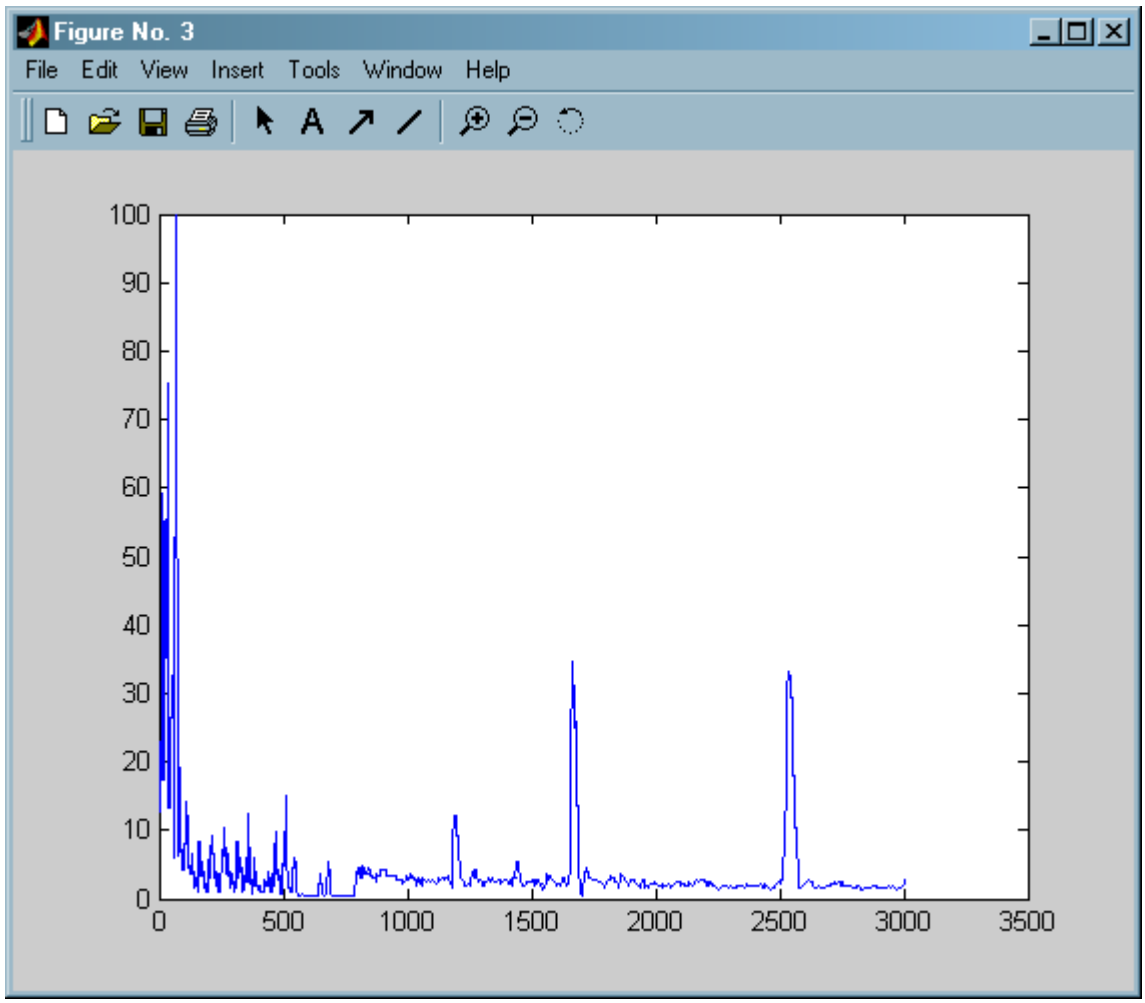
dove N è la dimensione della serie temporale,

4. si applica un algoritmo a soglia , che può essere di due tipi:

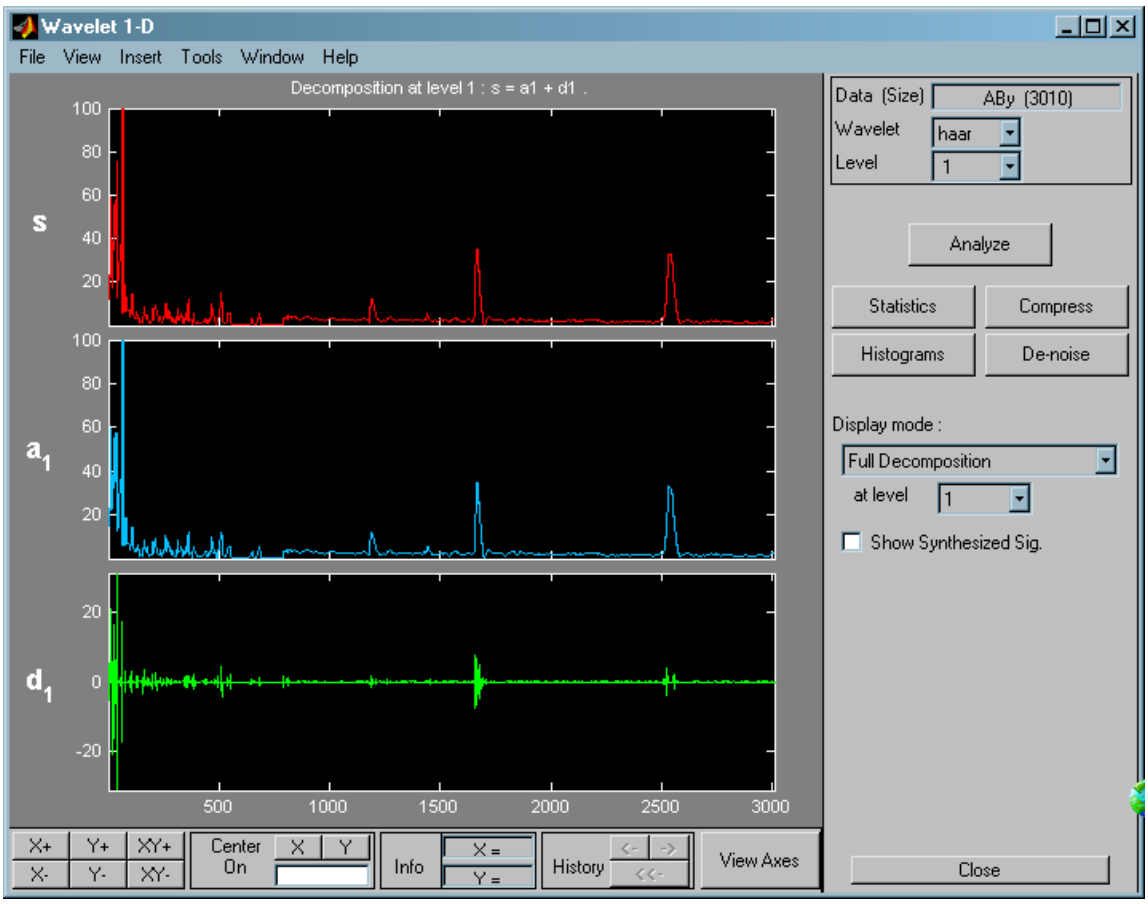
- hard thresholding, in cui ogni coefficiente minore della soglia viene azzerato
- soft thresholding, in cui i coefficienti minori della soglia vengono azzerati, mentre agli altri si sottrae la soglia stessa.

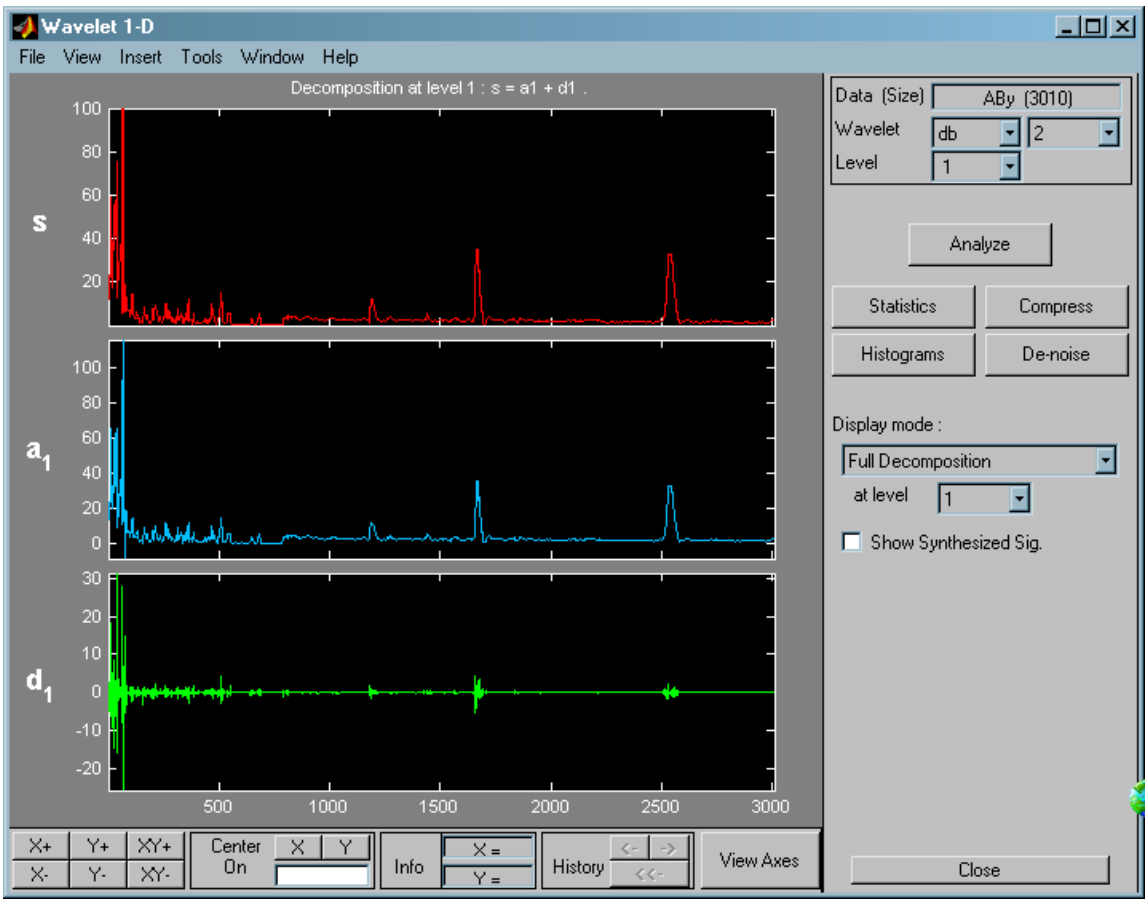
Par 4.13 Preprocessing e trasformata Wavelet

Tornando al problema del filtraggio degli spettri di massa batterici, rivedendo le informazioni raccolte si è cercato il tipo di Wavelet più adatto al caso. Si è utilizzato il Matlab per queste valutazione ed in realtà tutte le Wavelet del Matlab sono risultate abbastanza efficaci. Ad esempio si riporta lo studio sullo spettro dell Acinetobacter Baumanni. La figura seguente rappresenta lo spettro non elaborato di questo batterio.



Tale spettro è stato elaborato con il paccheto Wavelet del Matlab con due diverse Wavelet: la Haar e la Daubechies2.


Wavelet 1-D
File View Insert Tools Window Help
Decomposition at level 1 : s = a1 + d1 .
s
a1
d1
100
80
60
40
20
0
-20
500
1000
1500
2000
2500
3000
Data (Size)
ABy (3010)
Wavelet
haar
Level
1
Analyze
Statistics
Compress
Histograms
De-noise
Display mode :
Full Decomposition
at level
1
Show Synthesized Sig.
X+
Y+
XY+
X-
Y-
XY-
Center On
X
Y
Info
X =
Y =
History
View Axes
Close



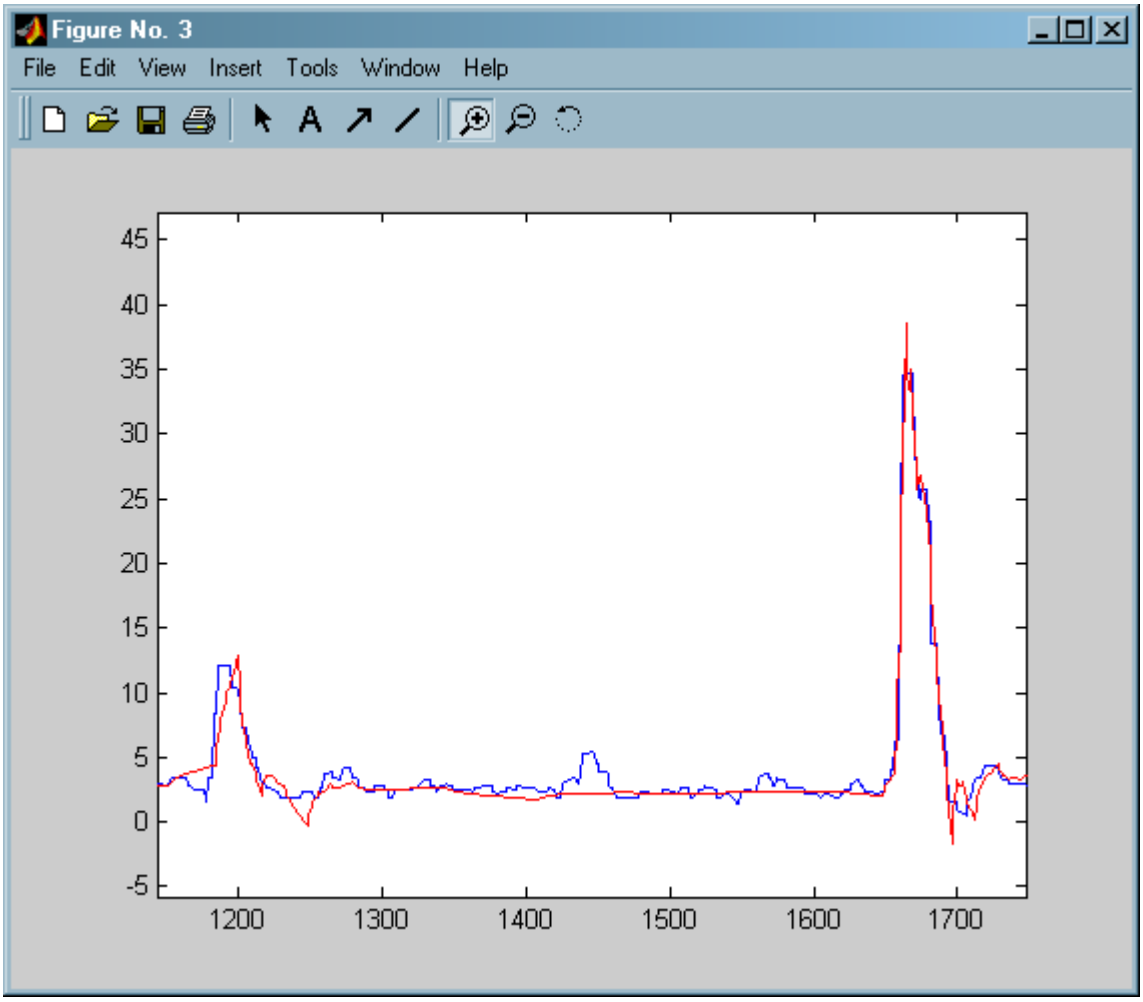
Lo spettro quindi è stato sintetizzato dalle sue due trasformate, confrontando i valori di queste antitrasformate rispetto allo spettro di partenza. La schermata del Matlab ha fornito questi risultati:

```
>> max(abs (ABytrasp - AByDB2))

ans =

  3.6674e-011

Import Wizard created variables in the current workspace.
>> max(abs(ABytrasp - AByHaar))

ans =

  2.8422e-014
```

Quindi la differenza tra l'errore con la trasformata Haar e con la trasformata Daubechies2 è dell'ordine delle migliaia, 10^{-11} contro 10^{-14} . Il valore assoluto di questi errori è però molto basso e la scelta fra le due trasformate è stata ritenuta non importante.
Il Matlab presenta delle ottimizzazioni degli algoritmi che possono aver influito sull'efficacia delle antitrasformazioni. Infatti le stesse valutazioni sono state fatte utilizzando degli algoritmi Wavelet realizzati in Java che appartengono al pacchetto Jsci e i risultati sono stati completamente diversi. Il pacchetto Jsci è un insieme di classi Java, disponibili in open-source, scritte da vari esperti per l'elaborazione in ambito scientifico; esso è una sorta di Matlab, scritto in Java. La scelta, allora, si è basata, oltre che sull'accuratezza, sui tempi di elaborazione, poiché gli spettri da elaborare in preprocessing sono molti e con un gran numero di punti ciascuno (dell'ordine delle migliaia). Le trasformate disponibili, scritte in Java, sono state fornite dal pacchetto JSci. Altre scelte hanno riguardato il filtraggio. Per il filtraggio si è usato un approccio diverso da quello proposto da Percival e Walden, poiché non sono note le caratteristiche del rumore e non è ben chiaro cosa possa essere considerato rumore e cosa possa essere considerato segnale. La scelta della soglia, quindi, è stata lasciata all'operatore, permettendogli di scegliere il miglior filtraggio secondo le sue conoscenze. La soglia è scelta come una percentuale del massimo valore dei coefficienti: tutti i coefficienti minori di questo valore sono azzerati. Qundi utilizzando, i coefficienti rimasti, si valuta l'antitrasformata DWT.
Dopo queste considerazioni, si è cercata una trasformata adatta al caso e disponibile in linguaggio Java.
Nel JSci sono proposte varie trasformate DWT ed anche una FastDaubechies2(), utile nel caso si debba trasformare una gran quantità di dati. Esso è un algoritmo ottimizzato per il calcolo della DWT e dell'antitrasformata, basato sulle Daubechies2, che riduce i tempi dell'elaborazione rispetto alle altre trasformate proposte. Ad esempio, questa FastDaubechies2 presenta degli accorgimenti che sacrificano l'accuratezza, ma migliorano i tempi di calcolo come

- cicli for senza condizione sul termine di un array che viene gestita con un interruzione mediante la coppia try-catch
- l'uso di tipi float invece di double.

Questa velocità, desiderabile nel contesto di corposi spettri di massa, comporta dei problemi sulla perfetta ricostruibilità del segnale. In particolare si è notato che il segnale filtrato e ricostruito presenta delle distorsioni sull'ampiezza rispetto al segnale di partenza. Sempre in relazione allo spettro visto in precedenza dell' Acinetobacter Baumanni, sovrapponendo i due spettri, quello filtrato e quello grezzo e ingrandendo, si ricava:



Il segnale in rosso è quello filtrato, presenta poco rumore, ma mostra anche dei valori dei picchi alterati rispetto al segnale grezzo, in blu.
Queste distorsioni sono molto negative, poiché alterano il valore delle intensità delle massa isotopiche rilevate. Non è possibile, quindi, a valle di questo filtraggio, rilevare i picchi con il calcolo dei massimi locali. Però, considerando l'algoritmo utilizzato con la FFT per rilevare i picchi, in cui si cercava di individuare la posizione del picco sul segnale filtrato, ma si rilevava la sua intensità sul segnale originale, si è applicata l'idea alla preprocessing con Wavelet. Quindi dopo un filtraggio mediante Wavelet, si sono confrontati il segnale filtrato con il segnale originale per la rilevazione dei picchi e per costruire ancora una volta una peaklist. Riassumendo c'erano due scelte per il preprocessing basato su Wavelet:

1. filtraggio mediante Wavelet con trasformata accurata, seguito da una rilevazione dei picchi basato sul calcolo dei massimi locali sul segnale filtrato,

2. filtraggio mediante una trasformata veloce, ma approssimata, e rilevazione dei picchi con confronto fra i due segnali, filtrato e non.

Si è optato per il secondo meccanismo, in cui le valli del segnale filtrato indicano l'inizio e fine dei picchi e l'intensità è letta sul seganle non filtrato come nell'algoritmo proposto da [1] per la FFT.

Par 4.14 Preprocessing mediante algoritmo Pepex-like

Il pepex è un software utilizzato per l'estrazione dei picchi automatica da spettri di massa generati dal Maldi-Tof. Il pepex lavora su spettri proteici e ne rileva i picchi utilizzando una soglia ed adattando una Gaussiana ai picchi selezionati dalla soglia. Il pepex cerca di raggruppare i picchi che appartengono ad una massa mono-isotopoca centrando su alcuni di essi una campana Gaussiana regolabile: tutti i picchi che vengono inglobati da questa campana sono ritenuti far parte di un singolo isotopo. Il pepex funziona in quattro passi:

1. stima della baseline,
2. stima del rumore, che viene effettuata centrando sul segnale finestre consecutive di una certa ampiezza, scelta dall'operatore, e applicando un meccanismo a soglia in cui la soglia è una percentuale del massimo valore rilevato nella finestra relativa (un meccanismo a soglia mobile),
3. stima dell'intensità, ampiezza, posizione dei picchi, in cui viene applicata una Gaussiana (le cui caratteristiche sono scelte dall'operatore) su ogni picco filtrato dal passaggio precedente, individuando tutti i picchi che finiscono all'interno di questa campana come picchi di masse monoisotopiche,
4. stima del picco monoisotopico, che raccogliendo le stime di possibili picchi monoisotopici del passo precedente, confronta le intensità valutate con una funzione che valuta le intensità previste degli isotopi attesi.

Di questo algoritmo, polarizzato sull'individuazione delle proteine, è stato considerato il passo del filtraggio del rumore.

Par 4.15 Adattamentodell'algoritmo Pepex al preprocessing

Il filtraggio del rumore del pepex è abbastanza banale, ma presenta delle caratteristiche interessanti perché elimina dei picchi dallo spettro in modo 'locale'. In altri termini opera come se il rumore fosse presente in modo diverso e con diverse intensità in diverse zone dello spettro. Il modo di osservare uno spettro di massa da parte di un esperto è proprio locale, poiché egli osserva i picchi significativi per zone dello spettro. Infatti il pepex è un software molto utilizzato e con buoni risultati nel campo della bioinformatica per l'individuazione di proteine. Nel caso degli spettri di massa batterici, l'utilizzo del meccanismo del pepex per valutare il rumore può essere proficuo, anche perché realizza una dinamica di filtraggio diversa da quelle viste finora e ritagliata sugli spettri di massa. L'implementazione di questo algoritmo si articola nei seguenti passi:

a. scelta della dimensione di una finestra da applicare sullo spettro per l'analisi locale
b. disposizione contigua delle finestre sullo spettro in modo da comprenderlo tutto
c. utilizzo del valore percentuale scelto dall'operatore per calcolare in ogni finestra una soglia locale (rispetto al massimo di intensità dei picchi nella finestra)
d. eliminazione del segnale al di sotto della soglia in ogni finestra
e. rilevamento dei picchi, valutando semplicemente i massimi locali e generando, quindi una peaklist.

Par 4.16 Description

Lo stadio 'Description' ha il compito di estrarre dalla peaklist in uscita dallo stadio di preprocessing una descrizione numerica, concisa e consistente dello spettro di massa batterico. Ha due funzioni principali: l'estrazione di questa descrizione e la sintesi di questa descrizione. Quindi, dopo lo stadio di preprocessing, anche questo stadio opera una sintesi delle informazioni contenute in uno spettro di massa, per ridurne la dimensionalità e renderlo più gestibile.
L'estrazione di una descrizione dai dati è detta 'features extraction', poiché estrae dai dati le caratteristiche peculiari efficaci nella loro discriminazione. La sintesi della descrizione, invece, cerca di estrarre dalle caratteristiche individuate precedentemente, le informazioni più rappresentative dei campioni sottoposti ed

è detta 'features selection'. In sostanza la features extraction rileva le caratteristiche più importanti nei dati, mentre la features selection cerca di individuare delle ridondanze, costruendo l'insieme più piccolo di caratteristiche efficace nel rappresentare i campioni. La descrizione di queste caratteristiche è detta 'features vector'.
La features extraction è relativa al dominio applicativo della classificazione e gli algoritmi per estrarre le caratteristiche dai campioni sono molto varie e dipendono dalle valutazioni di esperti del settore.
Le caratteristiche che permettono di discriminare un campione da un altro possono essere numerose e molto complicate da estrarre dai dati. Il campo della features extraction è molto vario e rappresenta un settore di ricerca a se stante in ogni dominio applicativo dell'intelligenza artificiale. Una features extraction efficace influenza in modo decisivo le prestazioni del classificatore. La features selection, invece, opera basandosi su valutazioni statistiche, elaborando i features vector estratti da tutto il dataset. Da queste valutazioni si individuano le caratteristiche migliori per descrivere i dati. Quindi, in uscita dallo stadio Description, vi è una rappresentazione ridotta, compatta ed efficace dei dati, che permette al classificatore di operare con migliore efficienza e migliore accuratezza. Questo è particolarmente vero per campioni la cui dimensionalità è molto elevata.
Lo stadio Description agisce in due momenti distinti; in un primo momento, estrae tutti i features vector da tutto il dataset e seleziona le caratteristiche non ridondanti da questi; in un momento successivo, utilizza le informazioni sulla selezione delle caratteristiche valutate sul dataset per costruire i features vector dei nuovi campioni che vanno classificati.
La prima è una fase di configurazione dello stadio Description e dipende dal dataset disponibile, dagli algoritmi scelti per l'estrazione e la selezione, mentre la seconda,invece, è una fase di piena operatività e di applicazione dei risultati ottenuti nella prima.
Nel caso della classificazione batterica, sono stati proposti diversi metodi per la features extraction, cioè per estrarre le caratteristiche dagli spettri in modo da renderle disponibili al classificatore. L'obiettivo principale di tutti questi metodi, è di ridurre la dimensionalità degli spettri, conservando le informazioni significative per la loro individuazione. Come precedentemente accennato, il numero di punti degli spettri di massa batterici e le scarsa disponibilità di spettri per ogni classe discriminabile implicano prestazioni poco accurate del classificatore. Quindi, in questo settore, come in quello dell'individuazione delle proteine, si cerca di ridurre in modo consistenze la presenza di ridondanze nei dati. Sono stati proposti in letteratura molti metodi, tra cui si cita (melissa

mcdermott), che propone l'uso di Wavelet per rappresentare gli spettri, e non solo per filtrarne il rumore.
In questo caso si è utilizato un altro approccio, che cerca oltretutto di risolvere un problema insito nelle misurazioni con il Maldi-Tof, che è l'errore di misura, dovuto in buona parte ad errori di calibrazione. Questo approccio è detto 'Calibrazione dei picchi'.

Par. 4.17: Calibrazione dei picchi

Dopo la fase di preprocessing, lo spettro consiste di una successione di possibili picchi di masse isotopiche. Considerando l'insieme degli spettri del dataset, si deve stabilire quali picchi appartengono alla stessa massa. In altri termini si devono trovare, tra tutti i picchi rilevati nel dataset, quali di essi appartengono allo stesso valore m/z. Inoltre si deve considerare, come per ogni strumento, un errore di misura. Questo errore può essere dovuto a varie cause ed in sintesi può essere valutato tra circa ± 0.03% e ±0.05% della massa misurata. L'errore deve essere considerato quando, valutando le massa delle peaklist, si cerca di stabilire se due picchi sono da considerarsi uguali o meno. Partendo quindi dalle liste ottenute dalla fase di preprocessing, si devono ottenere dei features vector che descrivono in modo uniforme e consistente tutti i campioni. Si raggruppano in una singola variabile del feature vector tutti i picchi che corrispondono ad una certa massa m/z. Due picchi i cui valori di m/z hanno una differenza minore del doppio dell'errore di misura, vengono unificati, più precisamente, due masse, m/z_a ed m/z_b , sono considerate identiche se i corrispondenti intervalli [$m/z_a \pm m/z_a * m_{err}$] e [$m/z_b \pm m/z_b * m_{err}$] si sovrappongono. La scelta su cosa sia coincidente e cosa sia diverso in questa fase è molto delicata, poiché influenza la rilevazione dei biomarkers significativi, determinando le prestazioni del classificatore. Per detrminare quali picchi appartengono alla stessa massa, si applica una procedura di raggruppamento gerarchico **[11]**, basata solo sui valori m/z rilevati. Questa procedura va comunque adattata al problema degli spettri aggiungendo ulteriori vincoli. Innanzitutto deve essere definita una misura della distanza tra due masse, poiché ogni algoritmo di raggruppamento(clustering) si basa su una qualche nozione di distanza tra le istanze che devono essere raggruppate. Si esprime la distanza in termini della scala delle masse, in modo che sia direttamente comparabile con l'errore m_{err} . Quindi la distanza tra le masse m_1 ed m_2 è definita come:

$$d(m_1, m_2) = \frac{|m_1 - m_2|}{\mu}, \ \mu = (m_1 + m_2)/2.$$

dove la distanza risulta espressa in funzione della loro media μ, che ha la stessa scala di m_{err} . Per completare la definizione delle caratteristiche di un algoritmo di raggruppamento gerarchico, si deve fornire una misura della distanza tra insiemi di instanze, cioè di insiemi di masse, in questo caso. A tal scopo si rappresentano due gruppi di masse (due clusters) C_1 e C_2, semplicemente con la media delle masse presenti in questi clusters e la distanza tra i due clusters risulta, quindi, la distanza tra queste medie:

$$d(\mu_{C_1}, \mu_{C_2})$$

Si sta cercando di realizzare,quindi, un clustering gerarchico basato sul cosiddetto *centroid linkage* . L'algoritmo comleto è illustrato nella seguente figura:

Algorithm 1 MassCluster(L)

{L : list of masses m_i from all the spectra}
$C_i \leftarrow m_i, m_i \in L$
$C \leftarrow \{C_i\}$
while Exist $C_l, C_k, \in C$ with $d(\mu_{C_l}, \mu_{C_k}) \leq 2 \times m_{err}$ AND
$argmax_{m_l, m_k} d(m_l, m_k) \leq 2 \times m_{err}$, $m_l \in C_l$, $m_k \in C_k$ AND
$d(\mu_{C_l}, \mu_{C_k}) = argmin_{C_i, C_j}(d(\mu_{C_i}, \mu_{C_j}))$ **do**
  merge(C_l, C_k)
end while
return C

Si notano i vincoli aggiunti per la natura particolare del problema degli spettri di massa:

a) i cluster possono essere accorpati solo se la loro distanza è minore di m_{err} (prima condizione del ciclo while);
b) se due cluster sono stati identificati come possibili candidati al raggruppamento, cioè $d(C_1, C_2) \leq 2 \times m_{err}$, la fusione dei due cluster avverrà realmente, se i due elementi più lontani di questi due cluster hanno una distanza minore del doppio dell'errore $2*m_{err}$ (seconda condizione del ciclo while); infatti, dal momento che ogni cluster contiene piccole perturbazioni di una data massa, non devono essere raggruppati picchi che appartengono a diverse masse;
c) tra le possibili coppie di candidati per il raggruppamento, l'algoritmo sceglie quella con la minima distanza (terza condizione del ciclo while)

Nella figura seguente, si illustra, schematicamente il funzionamento dell'algoritmo; in questo caso l'algoritmo termina in 6 passi:

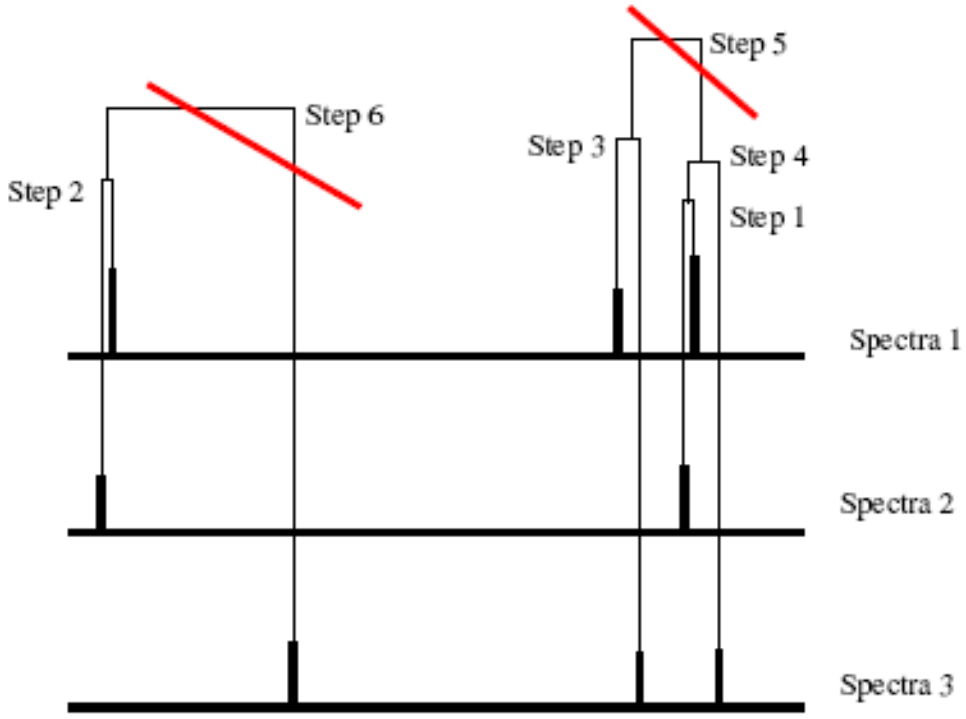


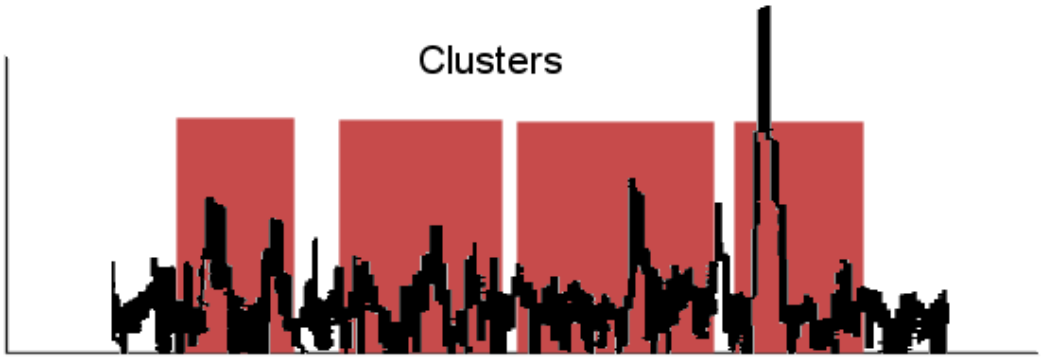
Fig. 1. Example of mass clustering. The numbered steps indicate the sequence of the merging steps. Steps 5 and 6 are not allowed because they violate domain constrains. Step 5 because it would put into the same cluster two masses that have a distance that is bigger than $2 \times m_{err}$; moreover it would have placed together two distinct masses from the same spectrum (spectrum 1), and step 6 because the distance of the two clusters exceeds the $2 \times m_{err}$ threshold.

Quando non vi sono più massa da raggruppare, l'algoritmo termina, restituendo una lista dei cluster, ciascuno dei quali corrisponde ad uno specifico rapporto m/z, dato dalla media delle masse trovate in esso. Quindi, ogni cluster diviene una feature per decrivere lo spettro e la lista dei cluster diviene un modo per costruire un feature vector per ogni spettro.

Par. 4.18: Implementazione della calibrazione dei picchi

Riassumendo, dopo lo stadio di preprocessing, che ha generato una lista ridondante di picchi caratteristici del campione, segue lo stadio di Description, il cui compito è di compattare le informazioni contenute nelle peak list e di costruire una rappresentazione compatta ed efficace dello spettro che possa essere elaborata dal classificatore. Per trasformare la peak list di uno spettro in features vector, lo stadio Description deve, quindi, operare su tutto il dataset, per configurare una struttura dati che gli permetta di trattare ogni peaklist sottoposta ad esso successivamente. Utilizzando, in particolare, l'algoritmo della calibrazione dei picchi visto precedentemente, si costruisce una lista di cluster che rappresentano i rapporti m/z rilevati nello spettro. L'implementazione pratica di questo algoritmo ha presentato non poche difficoltà. Innanzitutto, sono state raccolte delle peak list appartenenti a diversi campioni, costruite da esperti e derivate dallo stesso strumento. Da queste peak list sono state estratte tutte le masse rilevate, cioè tutti i rapporti m/z, tralasciando le intensità. Quindi si è trasferito l'algoritmo descritto in [PRA04.pdf] in una classe Java, senza particolari accorgimenti. Utilizzando tutti i valori delle masse m/z del dataset, si è avviata l'elaborazione del clustering. Nonostante la lista di masse non fosse corposissima,diverse migliaia di valori, confrontabile, quindi, con la dimensionalità di uno spettro grezzo, l'esecuzione dell'algoritmo è risultata particolarmente lenta. Su un P4, con ~ 392 MB di memoria, l'algoritmo impiega diverse ore a realizzare il clustering. Analizzando attentamente i passi del procedimento si è rilevato che, in sostanza, percorre la lista temporanea dei cluster più volte scegliendo ogni volta le coppie di cluster candidate. Per ogni cluster, percorre tutta la lista cercando di associare a questo cluster un altro per formare una coppia da fondere in un solo cluster, e questo per tutti i cluster della lista. Si è cercato quindi di evitare una ricerca completa su tutta la lista per ogni cluster. Ordinando i cluster della lista, i candidati per la fusione si susseguono e quindi la coppia di cluster per la fusione va scelta fra quelle costituite da cluster successivi. Infatti, solo i cluster successivi sono i più vicini, secondo la definizione di distanza utilizzata, e perciò è inutile accoppiare ogni cluster con tutti quelli della lista per verificare le condizioni. In sostanza si è cercato di ottimizzare l'algoritmo, sostituendolo con una versione più semplice che però richiede un previo ordinamento delle masse rilevate dal dataset. Utilizzando questo approccio, i tempi si sono ridotti notevolmente, realizzando la cluster list in pochi minuti. Si è verificata l'attendibilità di questo approccio confrontando i risultati dei due metodi con un dataset ridotto, per evitare tempi insostenibili di elaborazione. Entrambi i metodi hanno fornito gli stessi risultati.

In uscita a questa elaborazione vi è quindi una struttura dati che rappresenta i gruppi di picchi che rappresentano lo stesso ione. La struttura dati sui cluster è una matrice con diverse righe(una per ogni gruppo di picchi) e tre colonne, in cui la prima rappresenta il centro del cluster, mentre la seconda e la terza rappresentano l'inizio e la fine di un gruppo di picchi analoghi. La calibrazione dei picchi dipende dalla scelta dell'errore di calibrazione: si è scelto un errore dello 0.01 %, per valutarne gli effetti. Quindi una volta calcolata la struttura che descrive i cluster, ogni peaklist da sottomettere al classificatore va sottoposta al raggruppamento dei picchi per generare un features vector. Tale features vector ha una dimensionalità fissa, che è legata all'errore di calibrazione: più l'errore è basso più sono numerosi i picchi distinguibili in uno spettro. La dimensione costante del features vector è una proprietà desiderabile per la maggior parte dei classificatori. Con un errore dello 0.01 % e con il dataset fornito si sono rilevati 144 clusters e quindi un vettore delle features di 144 punti.
Ogni picco delle peaklist sottoposto all'estrazione di features viene considerato come appartenente ad un cluster o meno. Se più picchi appartengono ad uno stesso cluster viene considerata come intensità di picco quella del picco più elevato: il picco maggiore rappresenta l'abbondanza dello ione relativo al cluster.



Nella figura sono evidenziati i cluster in rosso e uno spettro. Dal passaggio di questo spettro attraverso questa elaborazione si ottiene un array di attributi(features) i cui valori sono il massimo dei picchi dello spettro negli intervalli dei clusters.

Par. 4.19: Classificazione

La classificazione è lo stadio più importante del sistema e presenta una gran varietà di scelte che ne possono determinare le prestazioni.

In letteratura sono state analizzate varie soluzioni per risolvere la classificazione di spettri di massa di qualsiasi genere, da quelli cosmici rilevati con particolari spettrografi, a quelli proteici rilevati con il Seldi-Tof o il Maldi-Tof. Molti di questi risultati sono applicabili anche agli spettri massa batterici. Come visto in precedenza, gli spettri di massa batterici hanno delle caratteristiche che favoriscono poco la loro classificazione. Ogni spettro ha una elevata dimensionalità e gli spettri rappresentativi di ogni batterio sono pochi in rapporto alle features estraibili da essi. Per addestrare efficacemente un classificatore il numero dei campioni rappresentativi di una categoria deve essere 5-10 volte superiore alle features discriminanti.

Par 4.20 Ricerche sulla classificazione (SVM, Bayes, reti neurali)

Anche dopo i primi due stadi di elaborazione, uno spettro può essere rappresentato da centinaia di variabili, mentre il numero di campioni disponibili per classe e di meno di una decina. Questa combinazione non garantisce una classificazione statisticamente significativa. In genere, in queste condizioni, il classificatore presenta delle prestazioni molto elevate, ma che si realizzano solo per il dataset utilizzato per l'addestramento; per altri campioni, il classificatore risulta inefficace. L'unica possibilità è di ottenere dal classificatore delle prestazioni rilevanti in diverse possibili configurazioni di training dataset e testing dataset realizzate in modo indipendente. Solo in questo caso il classificatore può considerarsi robusto.
Tuttavia, per la natura del problema degli spettri di massa, anche questo risultato può, nel concreto, essere non valido. Quindi, mentre una classificazione può essere ancora possibile, è difficile interpretare, per un operatore, i risultati ottenuti dall'addestramento del classificatore, in presenza di tutte queste features.
Un classificatore, in definitiva, deve essere robusto rispetto ad eventuali outliers, ed avere una grande capacità di generalizzazione. La robustezza rappresenta la capacità del classificatore di elaborare campioni sottoposti, anche se presentano delle anomalie sulle variabili del features vector. La capacità di generalizzazione implica l'abilità del classificatore di riconoscere campioni sconosciuti. La complessità del classificatore può variare moltissimo; i più semplici classificatori realizzano metodi di correlazione per valutare la similitudine tra campioni in ingresso e campioni memorizzati; i più complessi sono le Support Vector Machines non lineari. Questi esempi di classificatori, inoltre possono essere combinati in architetture complesse per discriminare tra campioni difficili da

prototipare. Tra queste architetture vi sono quelle dei classificatori multiesperto o i classificatori gerarchici. Quindi le possibilità di scelta per lo stadio di classificazione sono numerosissime, e la scelta può essere dettata da varie considerazioni. Le prestazioni dei classificatori, le risorse di calcolo disponibili, il dataset a disposizione ed anche le opinioni degli esperti del dominio applicativo sono gli elementi che determinano la scelta. In genere le scelte sui classificatori tendono verso i più complessi, senza effettivamente valutare il classificatore più adatto al dataset. Vi è un certo legame tra le caratteristiche di un dataset ed il classificatore più adatto; vi sono esempi di dataset molto semplici per i quali i classificatori più sofisticati hanno prestazioni relativamente povere.
Prima del progetto del classificatore , vanno fatte alcune considerazioni sul dataset. Il dataset va diviso in training set ed in test set. Il classificatore scelto viene ottimizzato sul training set, ma viene validato sul test set. Per evitare l'overfitting, si generano classificatori sub-ottimi, garantendo in questo modo un buon bilanciamento delle prestazioni sul training set e sul test set. Nel caso degli spettri di massa batterici, il dataset è sparso e questo approccio non è più utilizzabile. Si applica allora la crossvalidation (CV) al dataset, specificatamente un tipo di CV detta Leave One Out (LOO). La crossvalidation consiste, nella sua forma più elementare, nel dividere i dati in m sottoinsiemi di campioni. Ciascun sottoinsieme di campioni viene classificato utilizzando il modello costruito con gli altri m-1 campioni. Si addestra il classificatore con m–1 sottoinsiemi di campioni e si valuta l'errore di classificazione con l'emmesimo. Con tale procedura, si stima in modo efficiente l'errore. Infine si addestra il classificatore con tutti i campioni. In questo modo si possono valutare fenomeni di overfitting. La crossvalidation Leave One Out (LOO) è dovuta a Lachenbruch & Mickey e consiste in una crossvalidation con m uguale al numero dei campioni.
L'uso convenzionale della crossvalidation LOO, non produce,comunque, un classificatore robusto, ma soltanto ragionevolmente performante.
Ad ogni modo, la scarsità del dataset può portare ad un' alta performance del classificatore, sia con il training set, sia con il validator set, ma l'affidabilità di questi risultati è dubbia. Infatti, cambiando classificatore, si rilevano sempre risultati ottimi, che sono, oltretutto, indipendenti dalla riduzione delle features. Quindi, performance ottimali di classificazione, quasi indipendenti dal classificatore e dalla features selection, rappresentano una classificazione non rilevante statisticamente e con scarso potere di generalizzazione.
Per quanto riguarda la scelta del classficatore va anche considerata la scelta della features selection, infatti le prestazioni della classificazione sono influenzate dalla fase di features selection che la precede. Algoritmi di features selection possono essere utilizzati congiuntamente al classificatore o indipendentemente

da esso. In ogni caso vi sono coppie di algoritmi di selezione di features e classificatori che si adattano meglio di altre. Quindi la scelta del classificatore va fatta congiuntamente alla scelta dell'algoritmo di features selection.
Segue un esempio tratto da **[10]**, in cui si sono confrontati diversi dataset con diverse features selection e diversi classificatori. In questa tabella si può notare come variano le prestazioni di un classificatore, non solo in funzione del tipo di classificatore e della selezione delle features, ma anche in funzione del dataset:

Method	C 4.5 tree					
Dataset	Full set	FCBF	CorrSF	ReliefF	ConnSF	K-S CBF
Hypothyroid	**99.91**	*66.94*	85.92	98.94	93.45	98.68
Lung-cancer	*49.43*	63.87	**67.21**	63.87	63.22	64.76
Splice	94.35	94.05	*93.39*	**94.80**	93.63	94.05
Promoters	81.13	**85.84**	83.96	*65.09*	84.90	81.13
Average	81.21	*77.68*	82.62	80.68	83.80	**84.66**
Method	Naive Bayes					
Dataset	Full set	FCBF	CorrSF	ReliefF	ConnSF	K-S CBF
Hypothyroid	83.20	66.94	*58.48*	70.28	67.14	**85.83**
Lung-cancer	47.92	50.57	**73.48**	50.57	65.68	*41.74*
Splice	94.88	**96.03**	*93.31*	95.54	94.17	94.75
Promoters	90.56	**95.28**	93.39	*61.32*	93.39	87.35
Average	79.14	73.21	79.67	*69.43*	**80.10**	77.42
Method	1 Nearest Neighbor					
Dataset	Full set	FCBF	CorrSF	ReliefF	ConnSF	K-S CBF
Hypothyroid	*61.60*	83.89	71.72	78.92	**86.94**	83.93
Lung-cancer	39.94	*36.01*	**66.84**	53.13	65.41	45.70
Splice	*80.08*	84.45	**87.68**	84.04	86.77	83.82
Promoters	85.85	**92.45**	86.79	*59.43*	81.13	85.85
Average	*66.49*	74.28	78.26	69.01	**80.06**	74.83
Method	SVM					
Dataset	Full set	FCBF	CorrSF	ReliefF	ConnSF	K-S CBF
Hypothyroid	52.65	45.49	*44.07*	51.24	45.13	**84.31**
Lung-cancer	*41.37*	55.41	**66.07**	61.60	59.37	47.35
Splice	92.81	95.73	93.75	**95.75**	*90.08*	95.11
Promoters	**93.40**	91.50	77.36	*58.49*	87.33	93.11
Average	70.06	72.03	70.31	*66.77*	70.48	**80.04**

Table 4. Balanced accuracy for the 4 classification methods on features selected by each algorithm; bold face – best results, italics – worst.

Come si nota la differenza prestazionale tra le varie configurazioni può essere molto sensibile. Le possibili coppie di classificatori e selettori di features sono numerosissime e quindi una ricerca esaustiva e fuori dagli scopi di questo elaborato. In ogni caso si è proceduto alla valutazione dei possibili classificatori utilizzabili.

Par 4.21 Esperimenti con il dataset artificiale (dummy)

La scelta del classificatore e dell'architettura di classificazione è una fase fondamentale nella realizzazione del sistema. Innanzitutto la scarsità del dataset non permette di realizzare una valutazione esaustiva dei risultati. Come è stato illustrato, con un SFR troppo basso, i classificatori tendono ad avere tutti delle prestazioni notevoli numericamente, ma scarsamente valide statisticamente. Inoltre per ogni classificatore va scelta la features selection più adatta. In definitiva, si devono testare varie scelte possibili con un dataset ampio, per scegliere la configurazione migliore.
Il primo problema da risolvere è quello della scarsità del dataset. In genere per testare dei classificatori off-line,in fase di progetto del sistema, si usano dei dataset 'dummy',cioè dataset fantoccio, creati ad arte con caratteristiche simili a quelle del dataset originale. Questi dataset dummy servono a testare la robustezza del classificatore rispetto a certe variazioni dei campioni, in modo da scegliere il classificatore con maggiore potere di generalizzazione. In questo caso, oltre a testare il classificatore in varie condizioni, si è usato un dataset artificiale per aumentare la dimensione del dataset originale, in modo da avere risultati validi e confrontabili tra i vari classificatori. Le prestazioni dei classificatori, con un dataset dummy, vanno interpretate con attenzione, ma in ogni caso, si possono utilizzare diversi di questi dataset, con caratteristiche diverse, per scegliere il classificatore più adatto al problema in esame.
Nel caso degli spettri di massa batterici, si è cercato di capire il modo in cui variano gli spettri da un campione all'altro dello stesso ceppo batterico.
Nella figura seguente, tratta da (Batteri IGE 2004.pdf), in particolare, si nota la variazione dello spettro di una coltura al variare del tempo.
La stessa coltura è stata analizzata con il Maldi-Tof, immediatamente, poi conservata per 7 mesi e rianalizzata, e quindi conservata e rianalizzata dopo un anno.

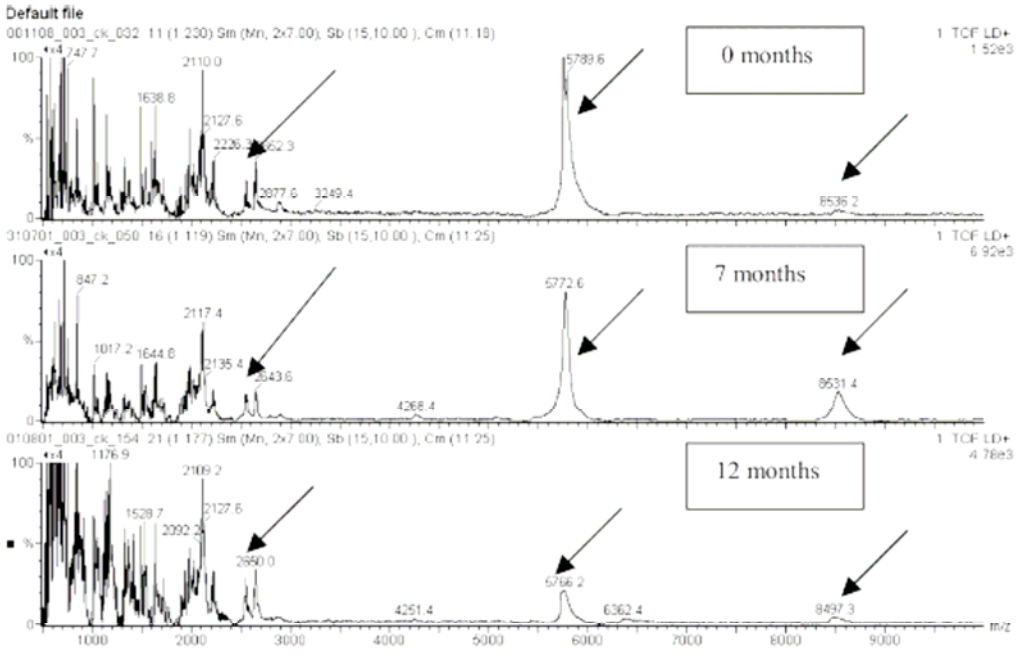


Fig. 2. Spectral profiles for the same strain of *A. calcoaceticus* (NCTC 7844) when prepared and analysed at three different time intervals: 0, 7 and 12 months. Key mass ions at ~2650, 5770 and 8500 Da are present in all three spectra, indicating the reproducibility of the profile over the 12-month period.

Si nota una variazione in ampiezza di alcuni picchi, ma non la loro posizione, cioè il valore m/z rimane costante, mentre l'intensità può variare.
Nella figura seguente, invece, si nota come possono variare gli spettri di un ceppo batterico da un laboratorio all'altro. In realtà, nella figura viene confrontato uno spettro ottenuto da un laboratorio con uno spettro dell' archivio NCTC relativo allo stesso ceppo batterico. Si nota una certa variazione nella configurazione di picchi omologhi negli spettri. Gruppi di picchi variano sia nelle intensità, analogamente al caso precedente, sia nei valori di m/z. In sostanza i gruppi di picchi possono anche traslare da una coltura ad un'altra.

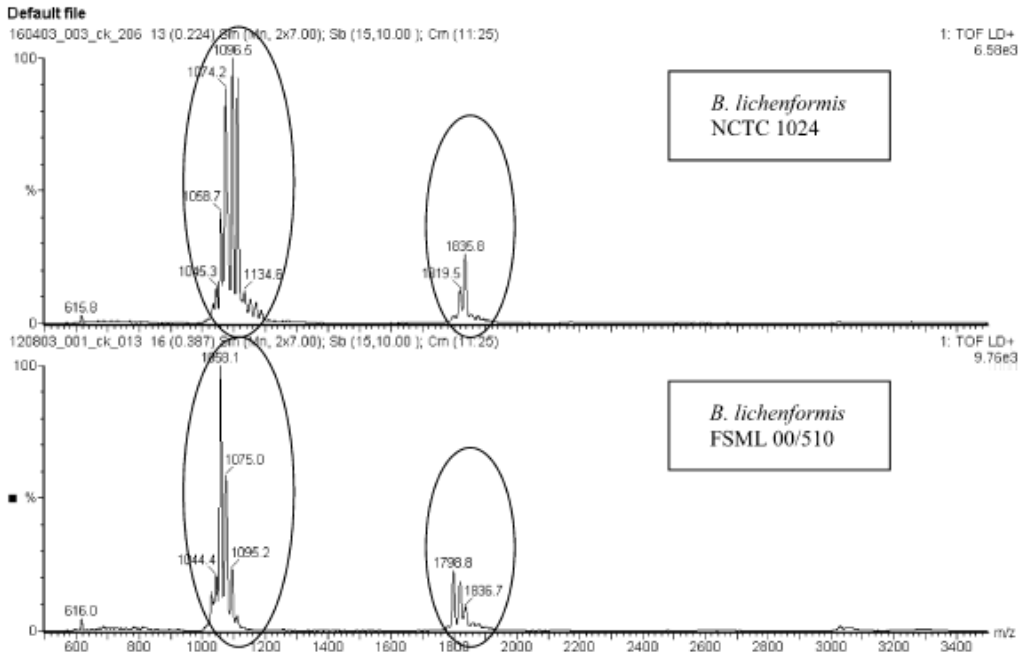


Fig. 3. Comparison of MALDI-TOF mass spectral data for a clinical isolate of *B. lichenformis* 00/0510 obtained from the Food Safety Microbiology Laboratory (FSML), HPA, London with the NCTC reference strain match from the database. The probability of the match being correct was 99.98%.

Nell'ultima figura, si confrontano spettri relativi ad ceppo, ottenuti nelle stesse condizioni, ma in tre laboratori diversi, con tre strumenti diversi, da tre operatori diversi. Si notano tutti i tipi di variazioni viste, ed inoltre anche una prevedibile variazione dell'ampiezza del rumore.

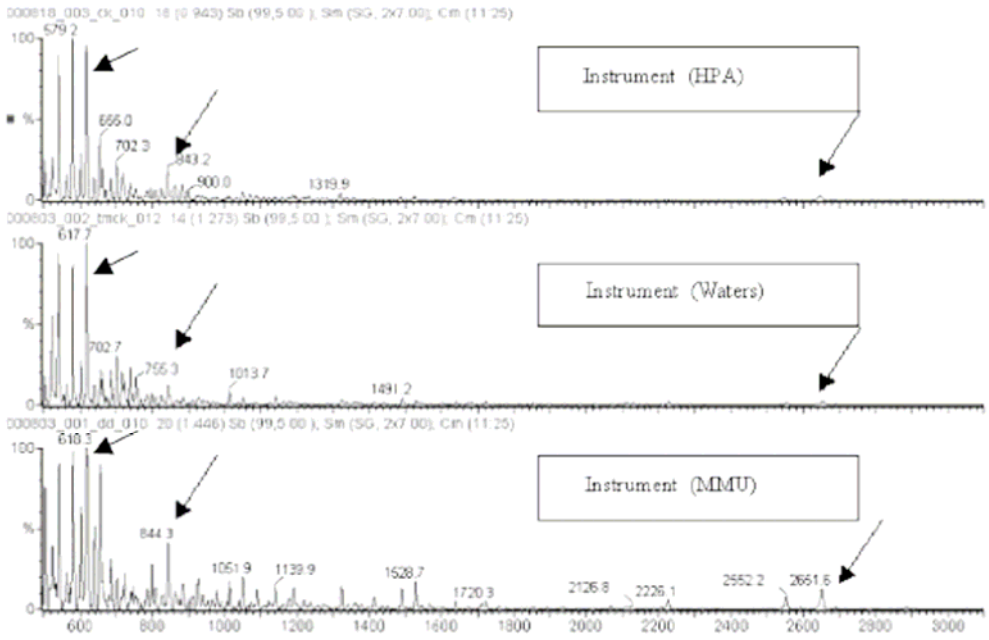


Fig. 4. A panel of strains was analysed at three different centres on three different instruments by different operators. This figure shows the spectral profiles *B. fragilis* NCTC 9343 and shows its reproducibility with key clusters of mass ions at ~620, 840 and 2650 Da.

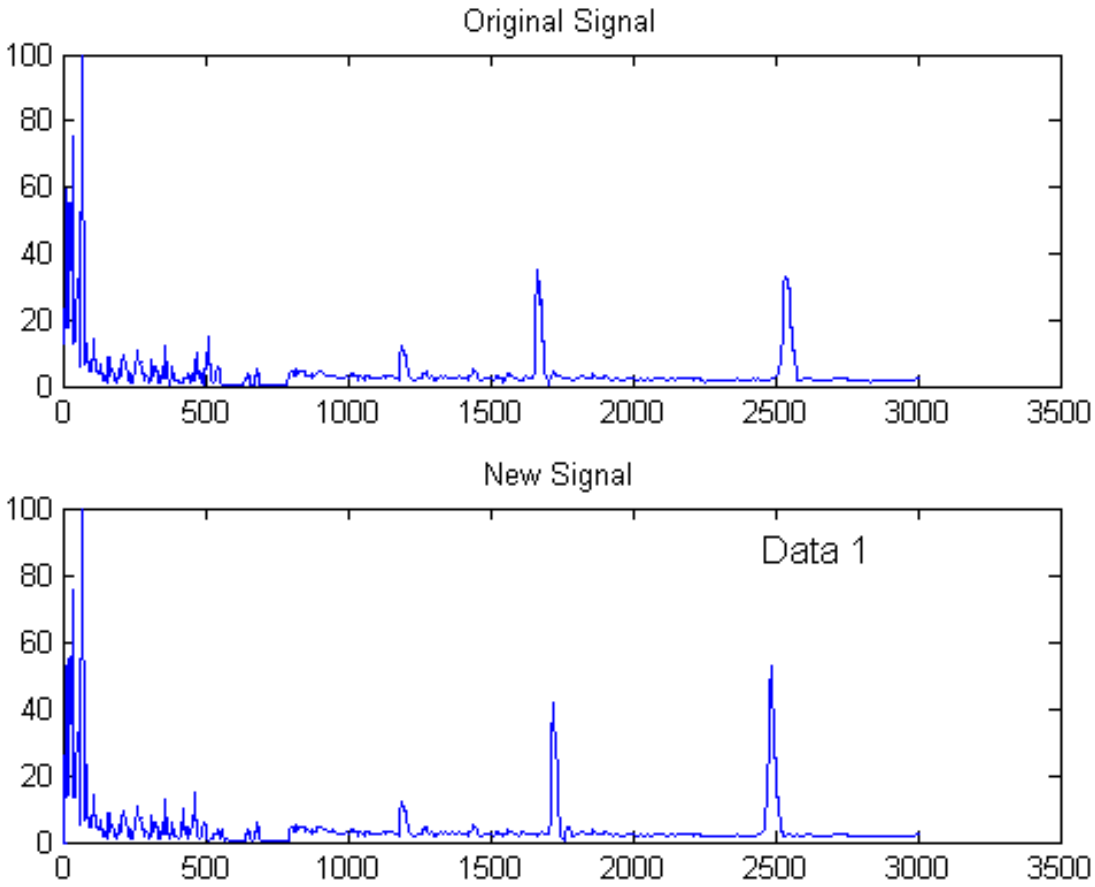
Ricapitolando, gli spettri di una stessa coltura batterica variano per:

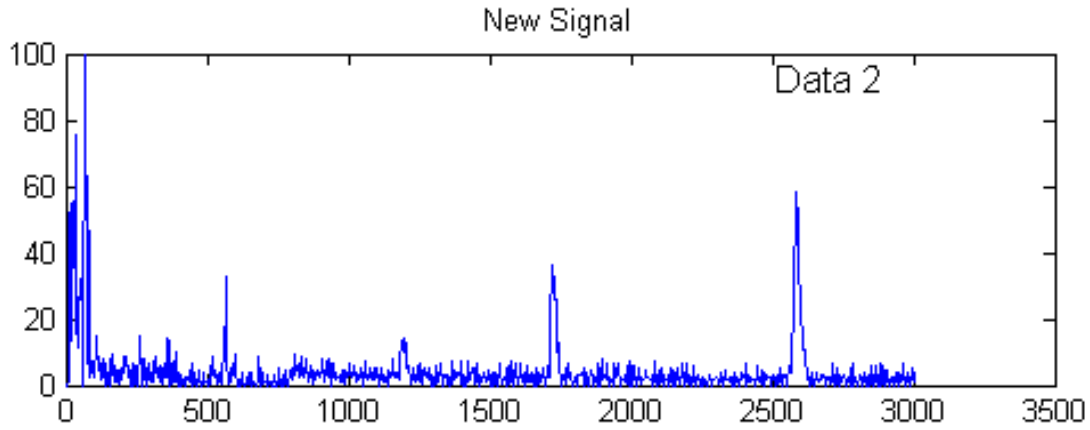
- traslazione di qualche picco significativo sull'asse m/z
- variazione dell'intensità di qualche picco, al limite annullamento di qualcuno di essi o emersione dal rumore
- variazione dell'intensità del rumore sovrapposto.

Per realizzare, quindi il dataset dummy, si è realizzata una classe Java, DataSetGen, che, con la scelta di alcuni parametri, genera un dataset artificiale riportando su alcuni spettri di partenza delle variazioni simili a quelle individuate precedentemente. Per quanto riguarda il rumore, un'istanza di questa classe sceglie casualmente di aggiungere del rumore o meno allo spettro e l'ampiezza massima di questo rumore e indicata dall'operatore. Per realizzare la traslazione dei picchi, DataSetGen sceglie dei picchi significativi, cioè oltre una soglia, casualmente e li trasla in modo anch'esso casuale entro una traslazione massima; l'operatore sceglie sia il livello della soglia, che la traslazione massima. La variazione in ampiezza è realizzata mediante l'individuazione, casuale, di picchi oltre una soglia e la loro diminuzione o incremento, scelti in modo random. Anche in questo caso l'operatore sceglie il massimo della variazione utilizzabile.
Con questo meccanismo è stato creato un dataset dummy da tredici spettri noti, costituito da dieci spettri diversi per ogni ceppo, per un totale di 130 spettri. Gli spettri di partenza sono realtivi a

1) Acinobacter baumanni
2) Bacillus pumilus
3) Burkholderia cepacia
4) Campylobacter coli
5) Citrobacter koseri
6) Chryseobacterium gleum
7) Escherichia coli (2 ceppi diversi)
8) Klebisiella pneumoniae
9) Neisseria gonorrhoeae
10) Rhodococcus equi
11) Shigella boydii
12) Staphylococcus aureus

Nelle figure seguenti si illustrano le modifiche su uno spettro, che hanno portato a generare due altri spettri, con il secondo alterato da rumore:





La scelta dei parametri per la generazione del dataset, è stata realizzata effettuando diversi tentativi, fino alla costruzione di un insiemi di spettri, le cui variazioni non fossero esageratamente accentuate, ma in linea con gli spettri visti in letteratura. Un certo grado di approssimazione in questa scelta è stato considerato utile per un test 'sovraddimensionato' dei classificatori.
Segue il codice datasetgen.java:

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

class readMassSpectrum
{

private String fileName;

//constructor
readMassSpectrum(String fN)
{
fileName = fN;
}


//calculate length of a file

int fileLength() throws IOException
{
FileReader fr = new FileReader( fileName );
BufferedReader in = new BufferedReader( fr );
```

```
String line;
int i = 0;

line = in.readLine();
while ( line != null )
{
i++;
line = in.readLine();
}
in.close();
return i;
}

//read a file into an array
double[][] go(int lnf) throws IOException
{

FileReader fr = new FileReader( fileName );
BufferedReader in = new BufferedReader( fr );

double[][] x=new double[lnf][2];

for(int i=0; i<lnf; i++)
{
StringTokenizer tok =
new StringTokenizer( in.readLine(), " " );

x[i][0]= Double.parseDouble( tok.nextToken().trim() );
x[i][1]= Double.parseDouble( tok.nextToken().trim() );


}
in.close();

/*
for(int j=0; j<lnf; j++)
{
System.out.println(x[j][0] +" "+ x[j][1] );
}
*/

return x;
}
}
```

```
//Class for traslation of groups of signal peaks

class peakTrasl
{
double th; // threshold to detect peaks
int dw; // range to detect peaks burst
int trAverage;  // traslation

//constructor
peakTrasl(double threshold,int dWindow,int traslation)
{
th=threshold;
dw=dWindow;
trAverage=traslation;
}

//traslation
void op(double[] in,Random rG)
{
//calculate window
int startWin=0;
int endWin=0;
int i=0;
int k;
int t=0;
boolean endSignal=false;
boolean foundW=false;
double[] temp;

while(!endSignal)
{

if(i==0 && in[i]>th) while(i<in.length && in[i]>th) i++;



if(i>=in.length)
{
endSignal=true;
break;
}

while(i<in.length && in[i]<th) i++;
```

```
if(i>=in.length)
{
endSignal=true;
break;
}

if(i-dw>0) startWin=i-dw;


foundW=false;

while(!foundW)
{
while(i<in.length && in[i]>th)i++;
if(i>=in.length)
{
endSignal=true;
break;
}
k=0;
while( i<in.length && k<dw && in[i]<th)
{
i++;
k++;
}
if(i>=in.length)
{
endSignal=true;
break;
}
if(k==dw)
{
endWin=i;
foundW=true;
}
}

//start traslation
if(!endSignal)
{
t=(int)Math.round( trAverage+3*(rG.nextGaussian()) ); //calculate traslation
temp=new double[t];
System.out.println("Traslation: " + t);
if(rG.nextBoolean())
{ System.out.println("Forward traslation");
//forward traslation
```

```
if(startWin-t>0 && endWin+t<in.length)
for(int j=0; j<t;j++)
{
temp[j]=in[endWin+j+1];
}
for(int j=0; j<endWin-startWin+1; j++)
{
in[endWin+t-j]=in[endWin-j];
}
for(int j=0; j<t;j++)
{
in[startWin+j]=temp[j];
}
i+=t;
}
}
else
{System.out.println("Backward traslation");
//backward traslation
if(startWin-t>0 && endWin+t<in.length)
{
for(int j=0; j<t;j++)
{
temp[j]=in[startWin-t+j];
}
for(int j=0; j<endWin-startWin+1; j++)
{
in[startWin-t+j]=in[startWin+j];
}
for(int j=0; j<t;j++)
{
in[endWin-t+1+j]=temp[j];
}
i+=t;
}
}

}
else System.out.println("End signal found");


}

}
}
```

```
class peakVar
{
double th;

//constructor
peakVar(double threshold)
{
th=threshold;
}
//peaks variation
void op(double[] in,Random rG,double A,int dG) throws IOException
{
int startPeak,endPeak;
double peak,intPeak;
boolean endSignal=false;
int i=0;
double m,c,s;

while(!endSignal)
{
if(i==0 && in[i]>th ) while( i<in.length && in[i]>th)i++;
if(i>=in.length)
{
endSignal=true;
break;
}

while( i<in.length && in[i]<th)i++;
if(i>=in.length)
{
endSignal=true;
break;
}
startPeak=i;

while( i<in.length && in[i]>=in[i-1])i++;
if(i>=in.length)
{
endSignal=true;
break;
}
intPeak=in[i-1];
peak=i-1;

while( i<in.length && in[i]>th)i++;
if(i>=in.length)
```

```
{
endSignal=true;
break;
}
endPeak=i;

//gaussian setting
m=peak;
s=endPeak-startPeak+2*dG;
//c=A*rG.nextGaussian();
c=2*rG.nextDouble();

//peak variation
if(endPeak+dG>=in.length)
{
endSignal=true;
break;
}
if(startPeak-dG<0)startPeak=dG;
for(int g=startPeak-dG; g<endPeak+dG; g++)
{
//in[g]=in[g] + c*Math.pow(Math.E, -(Math.pow(g-m,2)/s) );
in[g]=in[g]*c;
if(in[g]>100)in[g]=100;
if(in[g]<0)in[g]=0;
}
//System.out.println("Peak Variation: " + c +" between: " + startPeak + " and: " +
endPeak);
}
}
}


//Class for adding more noise to a signal

class Noise
{
double level;
double intensity;
double[] ns;
boolean set;

//constructor
Noise(double lv,double i)
{
level=lv;
```

```
intensity=i;
}
//create an array of noise
void noiseArray(int lng,Random rG) throws IOException
{
if(set)
{

double m=3;
double s=4.5;

//System.out.println("Array!");
try
{
ns=new double[lng];
double temp;

for(int i=3;i<lng-1;i+=7)
{
temp=(Math.pow(-1,i)*rG.nextDouble());

//temp*Math.pow(Math.E , -(Math.pow(i-m,2)/s) )

ns[i-3]=temp*Math.pow(Math.E , -(Math.pow((i-3)-i,2)/s) );
ns[i-2]=temp*Math.pow(Math.E , -(Math.pow((i-2)-i,2)/s) );
ns[i-1]=temp*Math.pow(Math.E , -(Math.pow((i-1)-i,2)/s) );
ns[i]=temp;
ns[i+1]=temp*Math.pow(Math.E , -(Math.pow((i+1)-i,2)/s) );
ns[i+2]=temp*Math.pow(Math.E , -(Math.pow((i+2)-i,2)/s) );
ns[i+3]=temp*Math.pow(Math.E , -(Math.pow((i+3)-i,2)/s) );

}
}catch(IndexOutOfBoundsException ex){};

}
}

//decide if add more noise
void addMore(Random rG)
{
set=rG.nextBoolean();
System.out.println("Set: " + set);
}
//work on signal with noise
void go(double[] signal)
{
```

```
double[] out=new double[signal.length];
double A;


if(set)
{ System.out.println("Noise with local max intensity: " + intensity);

for(int i=0;i<signal.length;i++)
if(signal[i]<level)
signal[i]=intensity*ns[i]+signal[i];


for(int i=0;i<signal.length;i++)
if(signal[i]<0)signal[i]=0;
}
}
}

public class DataSet
{
String fileName;
int nData;

//constructor
DataSet(String f,int n)
{
fileName=f;
nData=n;
}

void generate() throws IOException
{
int lnf;

readMassSpectrum rd=new readMassSpectrum(fileName);
lnf=rd.fileLength();

double[][] massSpectrum=new double[lnf][2];
double[] u=new double[lnf];

massSpectrum=rd.go(lnf);

for(int i=0; i<lnf; i++) u[i]=massSpectrum[i][1];

Random rndGen=new Random(100);//random numbers generator
double[] temp=new double[lnf];
```

```
//trasformation parameters

//traslation parameters
double pTrThreshold=10;  //threshold to detect peaks
int dW=50;    //range before and after the window to traslate in which there aren't peaks
int traslation=50;        //average traslation

//variation parameters
double pVarThreshold=10; //threshold to detect peaks
double amplitude=25;     //max amplitude of peak variation
int dG=50;  //range of variation influence before and after peak

//more noise parameters
double nThreshold=20;    //signal level under noise is added
double nIntensity=5;     //max intensity of noise
double nAmplitude;  //intensity of applied noise


for( int index=0; index<nData; index++)
{

for(int i=0; i<u.length;i++)temp[i]=u[i];

System.out.println("Index: " + index );

//peaks traslation
peakTrasl pT=new peakTrasl(pTrThreshold,dW,traslation);
pT.op(temp,rndGen);

//peaks variation
peakVar pV=new peakVar(pVarThreshold);
pV.op(temp,rndGen,amplitude,dG);

//Add more noise
nAmplitude=nIntensity*rndGen.nextDouble();
Noise nois=new Noise(nThreshold,nAmplitude);
nois.addMore(rndGen);
nois.noiseArray(lnf,rndGen);
nois.go(temp);


//write output
StringTokenizer nameFileOut=new StringTokenizer(fileName,".");
String baseName=nameFileOut.nextToken();
```

```
String fileNameOut = baseName + "_data2_" + index +  ".txt" ;
FileWriter writer = new FileWriter( fileNameOut );

for(int i=0;i<temp.length;i++)
writer.write(  massSpectrum[i][0] + " " + temp[i] +"\n"  );

writer.close();


}
}
}
```

Par 4.22 Esperimenti con i classificatori

Seguono le valutazioni e le statistiche degli addestramenti dei classificatori con i diversi tipi di filtraggio:

Metodo basato su FFT (classificatore: Naive Bayes)

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     bacteria
Instances:    130
Attributes:   426
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Naive Bayes Classifier

Class AB: Prior probability = 0.08

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

Time taken to build model: 0.18 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances	121	93.0769 %
Incorrectly Classified Instances	9	6.9231 %
Kappa statistic	0.925	
Mean absolute error	0.0107	
Root mean squared error	0.1032	
Relative absolute error	7.5 %	
Root relative squared error	38.7297 %	
Total Number of Instances	130	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0	1	1	1	AB
1	0	1	1	1	BC
0.8	0	1	0.8	0.889	BP
0.9	0	1	0.9	0.947	CC
1	0.008	0.909	1	0.952	CG
0.8	0.008	0.889	0.8	0.842	CK(!)
0.9	0	1	0.9	0.947	EC(!)
1	0.008	0.909	1	0.952	EC
1	0.008	0.909	1	0.952	KP
1	0.008	0.909	1	0.952	NG
1	0	1	1	1	RE
1	0.033	0.714	1	0.833	SA
0.7	0	1	0.7	0.824	SB

Metodo basato su FFT (classificatore: J48 (implementazione Weka dell'albero C4.5))

=== Run information ===

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: bacteria

Instances: 130
Attributes: 426
[list of attributes omitted]
Test mode: 10-fold cross-validation

=== Classifier model (full training set) ===

Time taken to build model: 1.16 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances	82	63.0769 %
Incorrectly Classified Instances	48	36.9231 %
Kappa statistic	0.6	
Mean absolute error	0.0584	
Root mean squared error	0.2279	
Relative absolute error	41.1481 %	
Root relative squared error	85.5203 %	
Total Number of Instances	130	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.5	0.025	0.625	0.5	0.556	AB
0.6	0.025	0.667	0.6	0.632	BC
0.5	0.017	0.714	0.5	0.588	BP
0.9	0.05	0.6	0.9	0.72	CC
0.7	0.025	0.7	0.7	0.7	CG
0.4	0.067	0.333	0.4	0.364	CK(!)
0.6	0.008	0.857	0.6	0.706	EC(!)
0.6	0.033	0.6	0.6	0.6	EC
0.8	0.033	0.667	0.8	0.727	KP
0.7	0.042	0.583	0.7	0.636	NG
0.8	0.042	0.615	0.8	0.696	RE
0.7	0.008	0.875	0.7	0.778	SA
0.4	0.025	0.571	0.4	0.471	SB

=== Confusion Matrix ===

```
 a b c d e f g h i j k l m   <-- classified as
 5 0 0 0 0 0 0 1 1 1 2 0 0 | a = AB
 0 6 1 0 1 0 0 0 1 0 1 0 0 | b = BC
 0 1 5 0 1 1 0 0 0 1 0 0 1 | c = BP
 0 0 0 9 0 1 0 0 0 0 0 0 0 | d = CC
 0 1 0 0 7 0 0 0 1 1 0 0 0 | e = CG
 0 0 0 3 0 4 1 1 0 0 0 1 0 | f = CK(!)
 0 0 0 1 0 2 6 0 0 0 1 0 0 | g = EC(!)
 0 0 0 0 1 2 0 6 1 0 0 0 0 | h = EC
 0 0 0 0 0 0 0 0 8 0 1 0 1 | i = KP
 0 1 0 0 0 1 0 0 0 7 0 0 1 | j = NG
 0 0 0 1 0 0 0 1 0 0 8 0 0 | k = RE
 1 0 0 0 0 1 0 1 0 0 0 7 0 | l = SA
 2 0 1 1 0 0 0 0 0 2 0 0 4 | m = SB
```

Metodo basato su FFT (classificatore: SMO (implementazione Weka della SVM))

```
=== Run information ===

Scheme:       weka.classifiers.functions.SMO -C 1.0 -E 1.0 -G 0.01 -A
250007 -T 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1
Relation:     bacteria
Instances:    130
Attributes:   426
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

SMO

Classifier for classes: AB, BC

BinarySMO

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

.....
.....

Time taken to build model: 18.7 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances	102	78.4615 %
Incorrectly Classified Instances	28	21.5385 %
Kappa statistic	0.7667	
Mean absolute error	0.1309	
Root mean squared error	0.2493	
Relative absolute error	92.1688 %	
Root relative squared error	93.5687 %	
Total Number of Instances	130	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0.042	0.667	1	0.8	AB
0.7	0.017	0.778	0.7	0.737	BC
0.9	0.025	0.75	0.9	0.818	BP
0.8	0.017	0.8	0.8	0.8	CC
0.9	0.033	0.692	0.9	0.783	CG
0.6	0.025	0.667	0.6	0.632	CK(!)
0.6	0	1	0.6	0.75	EC(!)
0.5	0.008	0.833	0.5	0.625	EC
0.8	0	1	0.8	0.889	KP
0.9	0.033	0.692	0.9	0.783	NG
1	0.025	0.769	1	0.87	RE
0.9	0.008	0.9	0.9	0.9	SA
0.6	0	1	0.6	0.75	SB

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	k	l	m	<-- classified as
10	0	0	0	0	0	0	0	0	0	0	0	0	a = AB
1	7	0	0	0	0	0	0	0	1	1	0	0	b = BC

```
 0  0  9  0  1  0  0  0  0  0  0  0  0 |  c = BP
 0  0  1  8  1  0  0  0  0  0  0  0  0 |  d = CC
 0  0  0  0  9  0  0  0  0  1  0  0  0 |  e = CG
 2  0  1  0  0  6  0  0  0  1  0  0  0 |  f = CK(!)
 0  0  0  0  0  3  6  1  0  0  0  0  0 |  g = EC(!)
 1  0  0  1  1  0  0  5  0  0  1  1  0 |  h = EC
 0  0  0  1  0  0  0  0  8  1  0  0  0 |  i = KP
 0  1  0  0  0  0  0  0  0  9  0  0  0 |  j = NG
 0  0  0  0  0  0  0  0  0  0 10  0  0 |  k = RE
 0  1  0  0  0  0  0  0  0  0  0  9  0 |  l = SA
 1  0  1  0  1  0  0  0  0  0  1  0  6 |  m = SB
```

Metodo basato su Wavelet (classificatore: Naive Bayes)

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     bacteria
Instances:    130
Attributes:   426
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Naive Bayes Classifier

Class AB: Prior probability = 0.08

Time taken to build model: 0.11 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         120               92.3077 %
Incorrectly Classified Instances        10                7.6923 %
Kappa statistic                          0.9167
Mean absolute error                      0.0118
```

Root mean squared error 0.1088
Relative absolute error 8.3336 %
Root relative squared error 40.8248 %
Total Number of Instances 130

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0	1	1	1	AB
1	0.017	0.833	1	0.909	BC
0.8	0	1	0.8	0.889	BP
1	0.008	0.909	1	0.952	CC
0.9	0	1	0.9	0.947	CG
0.9	0	1	0.9	0.947	CK(!)
1	0.008	0.909	1	0.952	EC(!)
0.9	0.008	0.9	0.9	0.9	EC
0.7	0.008	0.875	0.7	0.778	KP
0.9	0.008	0.9	0.9	0.9	NG
1	0	1	1	1	RE
1	0	1	1	1	SA
0.9	0.025	0.75	0.9	0.818	SB

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	k	l	m	<-- classified as
10	0	0	0	0	0	0	0	0	0	0	0	0	a = AB
0	10	0	0	0	0	0	0	0	0	0	0	0	b = BC
0	0	8	0	0	0	0	0	0	1	0	0	1	c = BP
0	0	0	10	0	0	0	0	0	0	0	0	0	d = CC
0	1	0	0	9	0	0	0	0	0	0	0	0	e = CG
0	1	0	0	0	9	0	0	0	0	0	0	0	f = CK(!)
0	0	0	0	0	0	10	0	0	0	0	0	0	g = EC(!)
0	0	0	0	0	0	0	9	0	0	0	0	1	h = EC
0	0	0	1	0	0	0	1	7	0	0	0	1	i = KP
0	0	0	0	0	0	1	0	0	9	0	0	0	j = NG
0	0	0	0	0	0	0	0	0	0	10	0	0	k = RE
0	0	0	0	0	0	0	0	0	0	0	10	0	l = SA
0	0	0	0	0	0	0	0	1	0	0	0	9	m = SB

Metodo basato su Wavelet (classificatore: J48)

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     bacteria
Instances:    130
Attributes:   426
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Time taken to build model: 1.4 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          92               70.7692 %
Incorrectly Classified Instances        38               29.2308 %
Kappa statistic                          0.6833
Mean absolute error                      0.0469
Root mean squared error                  0.2081
Relative absolute error                 33.0357 %
Root relative squared error             78.1136 %
Total Number of Instances              130

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   Class
                 0.9       0.017      0.818     0.9       0.857    AB
                 0.8       0.017      0.8       0.8       0.8      BC
                 0.5       0.042      0.5       0.5       0.5      BP
                 0.4       0.025      0.571     0.4       0.471    CC
                 1         0.017      0.833     1         0.909    CG
                 0.9       0.033      0.692     0.9       0.783    CK(!)
                 0.7       0.05       0.538     0.7       0.609    EC(!)
                 0.8       0.033      0.667     0.8       0.727    EC
```

```
 0.3      0.025     0.5      0.3      0.375    KP
 0.7      0.017     0.778    0.7      0.737    NG
 1        0.017     0.833    1        0.909    RE
 0.7      0.008     0.875    0.7      0.778    SA
 0.5      0.017     0.714    0.5      0.588    SB

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  i  j  k  l  m   <-- classified as
  9  0  0  0  0  0  0  0  1  0  0  0  0 |  a = AB
  0  8  0  0  0  1  0  0  0  1  0  0  0 |  b = BC
  0  0  5  1  0  1  2  0  0  0  1  0  0 |  c = BP
  0  1  2  4  0  0  0  2  0  0  0  0  1 |  d = CC
  0  0  0  0 10  0  0  0  0  0  0  0  0 |  e = CG
  0  0  0  0  0  9  1  0  0  0  0  0  0 |  f = CK(!)
  0  1  0  0  0  1  7  0  0  0  0  0  1 |  g = EC(!)
  0  0  1  0  0  0  1  8  0  0  0  0  0 |  h = EC
  1  0  0  1  0  1  1  1  3  1  0  1  0 |  i = KP
  0  0  0  0  1  0  0  0  1  7  1  0  0 |  j = NG
  0  0  0  0  0  0  0  0  0  0 10  0  0 |  k = RE
  1  0  1  0  1  0  0  0  0  0  0  7  0 |  l = SA
  0  0  1  1  0  0  1  1  1  0  0  0  5 |  m = SB
```

Metodo basato su Wavelet (classificatore: SMO)

```
=== Run information ===

Scheme:       weka.classifiers.functions.SMO -C 1.0 -E 1.0 -G 0.01 -A
250007 -T 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1
Relation:     bacteria
Instances:    130
Attributes:   426
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

=== Summary ===
```

Correctly Classified Instances	106	81.5385 %
Incorrectly Classified Instances	24	18.4615 %
Kappa statistic	0.8	
Mean absolute error	0.1312	
Root mean squared error	0.25	
Relative absolute error	92.4038 %	
Root relative squared error	93.8198 %	
Total Number of Instances	130	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0	1	1	1	AB
0.8	0.025	0.727	0.8	0.762	BC
0.9	0.058	0.563	0.9	0.692	BP
0.9	0.025	0.75	0.9	0.818	CC
1	0	1	1	1	CG
0.9	0.008	0.9	0.9	0.9	CK(!)
0.8	0	1	0.8	0.889	EC(!)
0.3	0.017	0.6	0.3	0.4	EC
0.8	0.025	0.727	0.8	0.762	KP
0.9	0.033	0.692	0.9	0.783	NG
0.9	0	1	0.9	0.947	RE
0.6	0.008	0.857	0.6	0.706	SA
0.8	0	1	0.8	0.889	SB

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	k	l	m	<-- classified as
10	0	0	0	0	0	0	0	0	0	0	0	0	a = AB
0	8	1	0	0	0	0	1	0	0	0	0	0	b = BC
0	0	9	0	0	0	0	1	0	0	0	0	0	c = BP
0	0	0	9	0	0	0	0	1	0	0	0	0	d = CC
0	0	0	0	10	0	0	0	0	0	0	0	0	e = CG
0	0	1	0	0	9	0	0	0	0	0	0	0	f = CK(!)
0	0	1	0	0	1	8	0	0	0	0	0	0	g = EC(!)
0	1	2	3	0	0	0	3	0	0	0	1	0	h = EC
0	0	1	0	0	0	0	0	8	1	0	0	0	i = KP

```
 0  1  0  0  0  0  0  0  0  9  0  0  0 |  j = NG
 0  1  0  0  0  0  0  0  0  0  9  0  0 |  k = RE
 0  0  1  0  0  0  0  0  1  2  0  6  0 |  l = SA
 0  0  0  0  0  0  0  0  1  1  0  0  8 |  m = SB
```

Metodo Pepex-like (classificatore: Naive Bayes)

```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     bacteria
Instances:    130
Attributes:   426
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

Naive Bayes Classifier

Time taken to build model: 0.11 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         114               87.6923 %
Incorrectly Classified Instances        16               12.3077 %
Kappa statistic                          0.8667
Mean absolute error                      0.0189
Root mean squared error                  0.1376
Relative absolute error                 13.3333 %
Root relative squared error             51.6398 %
Total Number of Instances              130

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   Class
                 0.8       0          1        0.8       0.889    AB
```

```
0.9      0        1        0.9      0.947    BC
0.8      0.033    0.667    0.8      0.727    BP
1        0        1        1        1        CC
0.9      0        1        0.9      0.947    CG
1        0        1        1        1        CK(!)
1        0        1        1        1        EC(!)
0.9      0.017    0.818    0.9      0.857    EC
0.8      0.025    0.727    0.8      0.762    KP
0.5      0        1        0.5      0.667    NG
0.9      0        1        0.9      0.947    RE
1        0.058    0.588    1        0.741    SA
0.9      0        1        0.9      0.947    SB

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  i  j  k  l  m   <-- classified as
  8  0  2  0  0  0  0  0  0  0  0  0  0 |  a = AB
  0  9  0  0  0  0  0  0  0  0  0  1  0 |  b = BC
  0  0  8  0  0  0  0  0  0  0  0  2  0 |  c = BP
  0  0  0 10  0  0  0  0  0  0  0  0  0 |  d = CC
  0  0  0  0  9  0  0  0  0  0  0  1  0 |  e = CG
  0  0  0  0  0 10  0  0  0  0  0  0  0 |  f = CK(!)
  0  0  0  0  0  0 10  0  0  0  0  0  0 |  g = EC(!)
  0  0  1  0  0  0  0  9  0  0  0  0  0 |  h = EC
  0  0  1  0  0  0  0  1  8  0  0  0  0 |  i = KP
  0  0  0  0  0  0  0  0  3  5  0  2  0 |  j = NG
  0  0  0  0  0  0  0  0  0  0  9  1  0 |  k = RE
  0  0  0  0  0  0  0  0  0  0  0 10  0 |  l = SA
  0  0  0  0  0  0  0  1  0  0  0  0  9 |  m = SB
```

Metodo Pepex-like (classificatore: J48)

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     bacteria
Instances:    130
```

Attributes: 426
[list of attributes omitted]
Test mode: 10-fold cross-validation

=== Classifier model (full training set) ===

Time taken to build model: 1.21 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances	107	82.3077 %
Incorrectly Classified Instances	23	17.6923 %
Kappa statistic	0.8083	
Mean absolute error	0.0283	
Root mean squared error	0.1626	
Relative absolute error	19.9167 %	
Root relative squared error	61.0361 %	
Total Number of Instances	130	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0	1	1	1	AB
0.9	0.025	0.75	0.9	0.818	BC
0.7	0.042	0.583	0.7	0.636	BP
1	0	1	1	1	CC
1	0.008	0.909	1	0.952	CG
1	0	1	1	1	CK(!)
1	0	1	1	1	EC(!)
0.7	0.042	0.583	0.7	0.636	EC
0.2	0.025	0.4	0.2	0.267	KP
0.8	0.008	0.889	0.8	0.842	NG
1	0.017	0.833	1	0.909	RE
1	0.008	0.909	1	0.952	SA
0.4	0.017	0.667	0.4	0.5	SB

=== Confusion Matrix ===

```
 a  b  c  d  e  f  g  h  i  j  k  l  m   <-- classified as
10  0  0  0  0  0  0  0  0  0  0  0  0 |  a = AB
 0  9  1  0  0  0  0  0  0  0  0  0  0 |  b = BC
 0  0  7  0  0  0  0  1  0  0  2  0  0 |  c = BP
 0  0  0 10  0  0  0  0  0  0  0  0  0 |  d = CC
 0  0  0  0 10  0  0  0  0  0  0  0  0 |  e = CG
 0  0  0  0  0 10  0  0  0  0  0  0  0 |  f = CK(!)
 0  0  0  0  0  0 10  0  0  0  0  0  0 |  g = EC(!)
 0  0  0  0  0  0  0  7  3  0  0  0  0 |  h = EC
 0  2  1  0  0  0  0  2  2  1  0  0  2 |  i = KP
 0  0  0  0  1  0  0  0  0  8  0  1  0 |  j = NG
 0  0  0  0  0  0  0  0  0  0 10  0  0 |  k = RE
 0  0  0  0  0  0  0  0  0  0  0 10  0 |  l = SA
 0  1  3  0  0  0  0  2  0  0  0  0  4 |  m = SB
```

Metodo Pepex-like (classificatore: SMO)

```
=== Run information ===

Scheme:       weka.classifiers.functions.SMO -C 1.0 -E 1.0 -G 0.01 -A
250007 -T 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1
Relation:     bacteria
Instances:    130
Attributes:   426
              [list of attributes omitted]
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

SMO

Time taken to build model: 16.71 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         116               89.2308 %
Incorrectly Classified Instances        14               10.7692 %
```

Kappa statistic 0.8833
Mean absolute error 0.1306
Root mean squared error 0.2487
Relative absolute error 91.9658 %
Root relative squared error 93.3377 %
Total Number of Instances 130

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0	1	1	1	AB
0.9	0.008	0.9	0.9	0.9	BC
1	0.05	0.625	1	0.769	BP
0.9	0.008	0.9	0.9	0.9	CC
1	0	1	1	1	CG
1	0.017	0.833	1	0.909	CK(!)
0.7	0	1	0.7	0.824	EC(!)
0.8	0	1	0.8	0.889	EC
0.8	0.017	0.8	0.8	0.8	KP
0.8	0.008	0.889	0.8	0.842	NG
1	0	1	1	1	RE
0.8	0.008	0.889	0.8	0.842	SA
0.9	0	1	0.9	0.947	SB

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	i	j	k	l	m	<-- classified as
10	0	0	0	0	0	0	0	0	0	0	0	0	a = AB
0	9	0	0	0	0	0	0	0	1	0	0	0	b = BC
0	0	10	0	0	0	0	0	0	0	0	0	0	c = BP
0	0	1	9	0	0	0	0	0	0	0	0	0	d = CC
0	0	0	0	10	0	0	0	0	0	0	0	0	e = CG
0	0	0	0	0	10	0	0	0	0	0	0	0	f = CK(!)
0	0	0	1	0	2	7	0	0	0	0	0	0	g = EC(!)
0	0	1	0	0	0	0	8	0	0	0	1	0	h = EC
0	0	2	0	0	0	0	0	8	0	0	0	0	i = KP
0	0	0	0	0	0	0	0	2	8	0	0	0	j = NG
0	0	0	0	0	0	0	0	0	0	10	0	0	k = RE
0	1	1	0	0	0	0	0	0	0	0	8	0	l = SA

0 0 1 0 0 0 0 0 0 0 0 0 9 | m = SB

Par 4.23 Considerazioni finali

Innanzitutto, l'addestramento della SVM andava fatto cercando di valutare la migliore coppia di parametri C e γ, ma si è scelta la coppia con i valori di default: ciò comporta una sottostima della prestazione di questa classificazione. A scopo di puro orientamento, questa scelta non compromette le valutazioni. Da un'ispezione delle statistiche risulta che il filtraggio Wavelet e la classificazione Naive Bayes risultano essere meglio performanti rispetto alle altre coppie. In ogni caso Naive Bayes dovrebbe essere il primo classificatore da considerare nella prova sugli spettri reali, poiché sembra essere il più adatto a questo problema di riconoscimento.

Capitolo 5

IL PROGETTO

Par 5.1: Premesse

Il progetto dell'applicazione segue le fasi classiche del processo di sviluppo di una applicazione sw qualsiasi:

1) raccolta e analisi dei requisiti
2) analisi dell'applicazione
3) progetto
4) implementazione
5) testing

Durante queste fasi, si è ricorsi ai costrutti dell'UML e allo standard IEEE Std 830-1993.

Par 5.2: Requisiti

I requisiti sono stati compilati riferendosi alle raccomandazioni per la specifica dei requisiti dello standard IEEE Std 830-1993.

Specifica dei requisiti dell'applicazione "Bacteria Classifier"

Cap1. Introduzione.

1.1 Scopo del documento

Questo documento definisce i requisiti dell' applicazione "Bacteria Classifier" per permetterne la specifica, il progetto e la validazione.
Il documento è indirizzato ai committenti ed agli sviluppatori dell' applicazione.

1.2 Obiettivi

"**Bacteria Classifier** " deve essere un'applicazione Web che permette la raccolta e la classificazione degli spettri di massa batterici ottenuti con lo spettrometro Maldi-Tof. "Bacteria Classifier" deve contenere una collezione di spettri di massa batterici, consultabile ed aggiornabile, e deve fornire un servizio di classificazione a supporto dell'individuazione di spettri batterici ignoti. L' applicazione deve permettere l'archiviazione di spettri di massa sia nella forma grezza (detta forma 'raw'), cioè nella forma in cui escono dallo strumento e senza trattamento, e sia nella forma di lista di picchi ('peak list'), cioè elaborati e rappresantati come una lista di picchi di massa significativi. Inoltre, "Bacteria Classifier" deve offrire la possibilità di estrarre picchi significativi da spettri raw in modo automatico e con parametri scelti dall'utente. I metodi per estrarre i picchi significativi sono tre:

- filtraggio del rumore con FFT seguito da localizzazione e quantificazione dei picchi; (vd. Capitolo 6)
- filtraggio del rumore mediante trasformata Wavelet discreta seguito da localizzazione e quantificazione dei picchi; (vd. Capitolo 6)
- filtraggio con metodo derivato dal Pepex e quantificazione dei picchi attraverso l'identificazione dei massimi locali. (vd. Capitolo 6)

Ogni spettro di massa archiviato è identificato da genere, specie e ceppo con eventuali note sulla coltura del batterio. "Bacteria Classifier" deve realizzare una stima sull'identità di spettri non noti, offrendo, o il genere, specie e ceppo ipotizzati o fornendo un grado di similitudine con gli spettri presenti in archivio. La stima sull'identità del batterio relativo ad uno spettro deve essere realizzata attraverso vari passaggi obbligatori:

- preprocessing: cioè l'estrazione dei picchi, la peaklist significativi se lo spettro è di tipo raw;
- features extraction: l'estrazione da una peaklist delle caratteristiche significative utili al suo riconoscimento, tali caratteristiche costituiscono il features vector;
- features selection: la selezione delle caratteristiche significative del features vector per la riduzione della dimensionalità delle stesse;
- classificazione: è lo stadio finale del processo di riconoscimento, che elabora il features vector dello spettro ignoto fornendo una stima dell'identità del batterio relativo.

Ogni classificatore, per essere utilizzabile, va adddestrato con un gruppo di features vectors derivati da tutti le peaklist disponibili, quindi un classificatore per poter riconoscere lo spettro un batterio, deve essere in condizione di 'generalizzare' le sue caratteristiche mediante diversi features vectors derivati da diversi spettri. L'insieme di tutti i features vectors che addestrano un classificatore è detto dataset.
Gli utenti dell'applicazione sono divisi in due categorie: gli utenti semplici e gli amministratori, che accedono all'applicazione fornendo 'user name' e 'password'.
Gli utenti hanno a disposizione alcune funzionalità base, mentre gli amministratori, oltre alle funzionalità base, possono gestire altri aspetti relativi all' applicazione.
Gli utenti semplici, per potere usufruire dei servizi, devono preventivamente iscriversi, fornendo alcuni dati personali, e la loro iscrizione viene confermata automaticamente.
Gli utenti semplici possono aggiungere uno spettro batterico di identità nota alla raccolta di "Bacteria Classifier", cercare uno spettro specifico e visualizzarlo, se presente.
Inoltre gli utenti semplici possono richiedere a "Bacteria Classifier" di esprimersi sull'identità di uno spettro di massa di un ceppo batterico non ancora identificato.
Gli amministratori dispongono di tutti i servizi forniti agli utenti semplici, ed inoltre possono interdire un utente, e gestire i meccanismi di classificazione.
"Bacteria Classifier" deve, in definitiva, facilitare l'archiviazione degli spettri di batteri e l'utilizzo di questo archivio; inoltre, deve essere di aiuto agli esperti nella valutazione dell'identità dei batteri analizzati in laboratorio con il Maldi-Tof.

1.3 Definizioni, acronimi e abbreviazioni.

"BC": "Bacteria Classifier".

Spettro di massa: Il diagramma o il segnale che riporta l'abbondanza di ogni ione in funzione del rapporto massa/carica.

Maldi-Tof: Strumento per la misura delle abbondanze di ogni ione in funzione del rapporto massa/carica.

Genere: Nelle scienze naturali il genere è una categoria che raggruppa le specie, in quanto aventi caratteristiche tra loro comuni.

Specie: Concetto base nella classificazione degli esseri viventi: la specie è l'unità tassonomica fondamentale.

Ceppo: Ulteriore suddivisione delle specie batteriche per caratteristiche particolari.

Coltura: Sviluppo di batteri su terreno composto da nutrienti adatti.

Spettro raw: Spettro di massa grezzo, come fornito da uno strumento per la spettrografia

Peaklist: Lista di picchi significativi che rappresentano le abbondanze caratteristiche di ioni in un certo spettro.

Features vector: Vettore delle caratteristiche significative di un campione adatto ad essere elaborato da un classificatore.

Dataset: Insieme di features vectors rappresentativo della realtà di interesse, utile all'addestramento del classificatore.

Visitatore: persona che accede a "Bacteria Classifier" e che può solo richiedere l'iscrizione;

Utente: persona iscritta e abilitata all'utilizzo dei servizi di ricerca, archiviazione e classificazione spettri;

Amministratore: utente privilegiato, che dispone dei servizi di utente ed anche di servizi di gestione di "Bacteria Classifier", come l'interdizione di un utente o la manutenzione dei meccanismi di classificazione.

1.4 Riferimenti

Bacteria Nomenclature (approved lists and validation lists) **[16]**.

1.5 Panoramica sul documento

Nel capitolo 2 di questo documento si descriverà l'applicazione "BC" in generale mentre nel capitolo 3 si descriveranno i requisiti ad un livello di dettaglio maggiore.

Cap2. Descrizione generale.

2.1 Prospettiva dell'applicazione

Sistema stand-alone che non si deve interfacciare con altri sistemi.

2.1.1 Interfacce di sistema

2.1.2 Interfacce utente
L'interfaccia verso l'utente deve essere a finestre con form da compilare.

2.1.3 Interfacce hardware

2.1.4 Interfacce software

2.1.5 Interfacce di comunicazione
Protocollo di comunicazione http.

2.2 Funzioni del prodotto

"BC" deve
- poter registrare nuovi utenti
- permettere o meno l'accesso agli utenti;
- permettere la ricerca di spettri dell'archivio;
- visualizzare i risultati della ricerca in archivio;
- permettere l'aggiunta di spettri all'archivio;
- cercare di classificare uno spettro sconosciuto;
- permettere la manutenzione dell'archivio e del classificatore.

2.3 Caratteristiche utente
L'utente sa navigare in Internet, sa compilare un Form e possiede un recapito di posta elettronica; inoltre conosce la tassonomia batterica e la spettrometria di massa batterica.

2.4 Vincoli

2.5 Assunzioni e dipendenze
Web container Apache Tomcat con supporto per Jakarta Struts e per database Mysql.

2.6 Dilazione di requisiti
Gestione più accurata degli aspetti del classificatore.

Cap3. Requisiti Specifici.

3.1 Interfacce esterne

3.2 Requisiti funzionali

3.2.1 Richiesta registrazione

3.2.1.1 Introduzione

Permette ad un visitatore di registrarsi per accedere ai servizi di "BC" offerti ad un utente semplice.

3.2.1.2 Input

Dati personali:

- nome*,
- cognome*,
- indirizzo,
- data* e luogo di nascita,
- professione,
- e-mail*,
- username scelto*,
- password scelta*.

L'asterisco indica i dati obbligatori.

3.2.1.3 Funzione

Visualizza un form per l'immissione di dati.
Controlla che i dati obbligatori siano stati digitati.
Verifica che l'indirizzo e-mail non appartenga ad un utente già registrato.

Verifica che username e password non siano già presenti.
Verifica che non vi sia interdizione sull'utente per eventi precedenti.

3.2.1.4 Output

Form per la digitazione dei dati personali.
Messaggio di dati obbligatori non digitati.
Messaggio di indirizzo e-mail appartenente ad un utente già registrato.
Messaggio di username e password già utilizzati.
Messaggio di richiesta respinta per interdizione utente.
Messaggio di richiesta inoltrata 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

Username e password entrambi obbligatori.

3.2.2.3 Funzione

Visualizza un form per l'immissione di dati.
Controlla che i dati obbligatori siano stati digitati.
Se tali dati non sono stati digitati visualizza nuovamente il Form ed il relativo messaggio di errore.
Verifica che username e password corrispondano ad un utente registrato o ad un amministratore.
Nel caso in cui username e password corrispondano ad un utente registrato o ad un amministratore, visualizza un menu differenziato per utente o per amministratore con i rispettivi servizi disponibili.
Nel caso in cui username e password non corrispondano ad utenti o amministratori, visualizza nuovamente il Form con il relativo messaggio di errore.
Verifica che non vi sia interdizione sull'utente, nel qual caso avverte l'utente con un messaggio e non gli permette l'accesso ai servizi dell'applicazione.

3.2.2.4 Output

Form per la digitazione dei dati personali.
Messaggio di dati obbligatori non digitati.
Messaggio di dati non corrispondenti ad alcun utente registrato o amministratore ed accesso negato.
Messaggio di accesso garantito.
Menu specializzati per utente o amministratore.
Messaggio di utente interdetto ed accesso negato.

3.2.3 Ricerca spettro nell'archivio.

3.2.3.1 Introduzione

Questa funzione ricerca uno spettro fra quelli presenti in archivio, utilizzando i dati immessi dall'utente o amministratore.

3.2.3.2 Input

Genere, specie e ceppo di un batterio di cui si vuole ricercare lo spettro.
Il genere di un batterio è obbligatorio.

3.2.3.3 Funzione

Visualizza un Form per la ricerca di spettri relativi ad un batterio.
Controlla che almeno il genere sia stato digitato.
Se il genere del batterio non è stato immesso visualizza un messaggio di errore e ripropone il Form.
Verifica che i dati immessi siano coerenti con la nomenclatura accettata, altrimenti visualizza un messaggio di errore e ripropone il Form.
Ricerca nel database gli spettri corrispondenti ai dati immessi dall'utente o dall'amministratore. Se presenti, raccoglie questi spettri, altrimenti visualizza un messaggio di ricerca senza esito.

3.2.3.4 Output

Form con genere, specie e ceppo di batterio.
Messaggio di genere non digitato.

Messaggio di nomenclatura non coerente con lo standard accettato.
Messaggio di ricerca senza esito.
Spettri letti dal database.

3.2.4 Visualizzazione spettri.

3.2.4.1 Introduzione

Funzione che permette la visualizzaione di spettri sia in forma raw che in forma peaklist.

3.2.4.2 Input

Spettro di massa e suo tipo (raw o peaklist).

3.2.4.3 Funzione

Visualizza uno spettro di massa.
Se lo spettro in ingresso è una peaklist ne disegna un grafico quotato e ne elenca i valori, mentre se lo spettro è raw ne visualizza solo un grafico quotato.

3.2.4.4 Output

Grafico quotato.
Elenco di valori presenti in uno spettro nella forma peaklist.

3.2.5 Estrazione peaklist con metodo FFT

3.2.5.1 Introduzione

Funzione che realizza una estrazione di peaklist da uno spettro raw, cercando di individuare i picchi significativi con il metodo basato su filtraggio mediante FFT.

3.2.5.2 Input

Spettro raw.
Frequenza di taglio.
(entrambi obbligatori)

3.2.5.3 Funzione

Visualizza Form per i dati in ingresso.
Controlla che lo spettro sia stato immesso, altrimenti visualizza un messaggio di errore.
Controlla che la frequenza di taglio sia stata digitata, altrimenti visualizza un messaggio di errore.
Verifica che la frequenza di taglio immessa abbia un valore compreso tra 1 e 1000, altrimenti visualizza un messaggio di errore.
Elabora lo spettro immesso con il metodo del filtraggio mediante FFT ed impiegando come frequenza di taglio il valore immesso dall'utente. Dopo il filtraggio, estrae i picchi significativi con il metodo[1].

3.2.5.4 Output

Form per i dati in ingresso.
Peak list
Messaggio di spettro non immesso.
Messaggio di Frequenza di taglio non digitata.
Messaggio di Frequenza di taglio non valida.

3.2.6 Estrazione peaklist con metodo Wavelet

3.2.6.1 Introduzione

Funzione che realizza una estrazione di peaklist da uno spettro raw, cercando di individuare i picchi significativi con il metodo basato sulla trasformata Wavelet.

3.2.6.2 Input

Spettro raw.
Soglia per i coefficienti Wavelet.
(entrambi obbligatori)

3.2.6.3 Funzione

Visualizza un Form per i dati in ingresso.
Controlla che lo spettro sia stato immesso, altrimenti visualizza un messaggio di errore.
Controlla che la soglia per i coefficienti sia stata digitata, altrimenti visualizza un messaggio di errore.
Verifica che la soglia abbia un valore compreso tra 0 e 1,

altrimenti visualizza un segnale di errore.
Elabora lo spettro immesso, filtrandolo con il metodo della trasformata Wavelet e ne estrae i picchi significativi mediante un sistema di valutazione di posizione e ampiezza picchi.

3.2.6.4 Output

Form per i dati in ingresso.
Peak list.
Messaggio di spettro non immesso.
Messaggio di soglia non digitata.
Messaggio di soglia non valida.

3.2.7 Estrazione peaklist con metodo Pepex-like

3.2.7.1 Introduzione

Funzione che realizza una estrazione di peaklist da uno spettro raw, cercando di individuare i picchi significativi con il metodo dedotto dal Pepex.

3.2.7.2 Input

Spettro raw.
Ampiezza della finestra.
Soglia per il rumore.
(tutti obbligatori)

3.2.7.3 Funzione

Visualizza un Form per i dati in ingresso.
Controlla che lo spettro sia stato immesso, altrimenti visualizza un messaggio di errore.
Controlla che l'ampiezza della finestra sia stata digitata, altrimenti visualizza un messaggio di errore.
Controlla che il valore di questa ampiezza sia un intero compreso tra 1 e la lunghezza dello spettro, altrimenti visualizza un messaggio di errore.
Controlla che la soglia per il rumore sia stata digitata, altrimenti visualizza un messaggio di errore.
Verifica che la soglia sia un numero reale compreso tra 0 e 1, altrimenti visualizza un segnale di errore.

Elabora lo spettro immesso, filtrandolo con il metodo Pepex-like e ne estrae i picchi significativi mediante un sistema di valutazione di massimi locali.

3.2.7.4 Output

Form per i dati in ingresso.
Peak list.
Messaggio di spettro non immesso.
Messaggio di ampiezza di finestra non digitata.
Messaggio di ampiezza di finestra non valida.
Messaggio di soglia non digitata.
Messaggio di soglia non valida.

3.2.8 Pulizia peaklist

3.2.8.1 Introduzione

Questa funzionalità permette ad un utente di attivare un ulteriore pulizia di una peaklist ricavata da uno spettro raw. Essa elimina dei picchi di bassa intensità, utilizzando una soglia globale per deciderne l'eliminazione.

3.2.8.2 Input

Peaklist
Soglia
(entrambi obbligatori)

3.2.8.3 Funzione

Visualizza un Form per i dati in ingresso.
Controlla che la soglia sia stata digitata, altrimenti visualizza un messaggio di errore.
Controlla che la soglia sia un numero reale tra 0 e 1, altrimenti visualizza un messaggio di errore.
Elabora lo spettro eliminando tutti i picchi la cui intensità sia inferiore ad una frazione del massimo delle intensità dello spettro.Tale frazione del massimo è calcolata con la soglia immessa.

3.2.8.4 Output

Form per i dati in ingresso.
Peaklist.
Messaggio di soglia non digitata.
Messaggio di soglia non valida.

3.2.9 Classificazione

3.2.9.1 Introduzione

Questa funzionalità calcola una stima dell'identità di uno spettro sconosciuto.

3.2.9.2 Input

Peaklist.
Scelta classificatore.
Classificatore.
(entrambi obbligatori)

3.2.9.3 Funzione

Visualizza un Form per i dati in ingresso.
Verifica che lo spettro sia stato immesso, altrimenti visualizza un messaggio di errore.
Visualizza i classificatori disponibili per il riconoscimento.
Legge il classificatore scelto.
Classifica lo spettro in ingresso utilizzando il classificatore scelto dall'utente.

3.2.9.4 Output

Form per la scelta del classificatore e caricamento della peaklist.
Messaggio di spettro non immesso.
Genere, specie e ceppo stimati.

3.2.10 Aggiunta spettro batterico all'archivio

3.2.10.1 Introduzione

Permette ad un utente di aggiungere alla collezione un nuovo spettro di massa batterico, ottenuto dall'utente.

3.2.10.2 Input

Genere*
Specie*
Ceppo
Note sulla coltura
Tipo di spettro, cioè raw o peaklist*
Spettro*
(con l'asterisco i valori obbligatori)

3.2.10.3 Funzione

Visualizza un Form per i dati in ingresso.
Verifica che i campi obbligatori siano stati digitati, altrimenti visualizza un messaggio di errore.
Valuta se genere, specie e ceppo sono coerenti con la nomenclatura batterica, altrimenti segnala un errore.
Se genere, specie e ceppo sono validi, aggiorna l'archivio con con lo spettro in ingresso e visualizza un messaggio sull'esito dell'operazione.

3.2.10.4 Output

Form per l'aggiunta di uno spettro all'archivio.
Messaggio di campo obbligatorio non compilato.
Messaggio di nomenclatura non valida.
Messaggio di avvenuta archiviazione.
Spettro.

3.2.11 Cerca utente

3.2.11.1 Introduzione

Questa funzione permette ad un amministratore di cercare un utente registrato.

3.2.11.2 Input

User name
(campo unico obbligatorio)

3.2.11.3 Funzione

Visualizza un Form per i dati in ingresso.

Controlla che lo user name sia stato digitato, altrimenti visualizza un messaggio di errore.
Cerca in archivio utenti l'utente corrispondente allo user name digitato. Se è presente un utente con questo user name, ne legge i dati dall'archivio. Se non vi è nessun utente visualizza un messaggio di ricerca senza esito.

3.2.11.4 Output

Form per i dati in ingresso.
Messaggio di errore di user name non digitato.
Messaggio di utente non presente.
Dati utente.

3.2.12 Visualizzazione utente

3.2.12.1 Introduzione

Funzione che visualizza i dati di un utente.

3.2.12.2 Input

Dati utente

3.2.12.3 Funzione

Costruisce un quadro di dettaglio con tutte le informazioni su un utente, per la consultazione da parte di un amministratore.

3.2.12.4 Output

Vista dati utente.

3.2.13 Interdizione utente

3.2.13.1 Introduzione

Funzione per porre un utente in interdizione, cioè all'utente scelto è impedito l'accesso all'applicazione.

3.2.13.2 Input

User name.

3.2.13.3 Funzione

Legge gli utenti registrati.
Visualizza un Form per la scelta di un utente fra quelli registrati.
L'utente prescelto è interdetto e qualsiasi suo accesso viene impedito.

3.2.13.4 Output

Form per la scelta di un utente.

3.2.14 Cerca Classificatore

3.2.14.1 Introduzione

Funzione per la visualizzazione dei classificatori disponibili in archivio.

3.2.14.2 Input

Dati classificatore

3.2.14.3 Funzione

Cerca tutti i classificatori disponibili in archivio, ne legge i dati e li visualizza.

3.2.14.4 Output

Schermata con dati sul classificatore.

3.2.16 Aggiunta Classificatore

3.2.16.1 Introduzione

Permette ad un amministratore di aggiungere un nuovo tipo di classificatore a quelli disponibili in archivio.

3.2.16.2 Input

Classificatore.
Dati classificatore.
(entrambi obbligatori)

3.2.16.3 Funzione

Visualizza un Form per i dati in ingresso.
Controlla che i dati obbligatori siano stati immessi, altrimenti visualizza un messaggio di errore.
Registra il classificatore e i sui dati in archivio.

3.2.16.4 Output

Form per l'aggiunta di un classificatore.
Messaggio di avvenuta archiviazione.
Dati e classificatore.

3.2.17 Addestramento Classificatore

3.2.17.1 Introduzione

Funzione per l'addestramento o l'aggiornamento di un classificatore con i features vectors presenti nell'archivio, cioè con il dataset disponibile in quel momento.

3.2.17.2 Input

Codice classificatore.

3.2.17.3 Funzione

Visualizza i classificatori disponibili per l'addestramento o l'aggiornamento.
Addestra o aggiorna un classificatore scelto con i features vectors presenti in archivio.

3.2.17.4 Output

Schermata con i classificatori disponibili.
Statistiche dell'addestramento.

3.2.18 Estrazione features vector

3.2.18.1 Introduzione

Funzione per l'estrazione da una peaklist delle caratteristiche utili alla sua classificazione, in un formato che sia accettabile da un classificatore. Tale funzione utilizza i dati ricavati da una

precedente calibrazione su tutto il dataset per individuare le features.

3.2.18.2 Input

Peaklist.
Cluster data.

3.2.18.3 Funzione

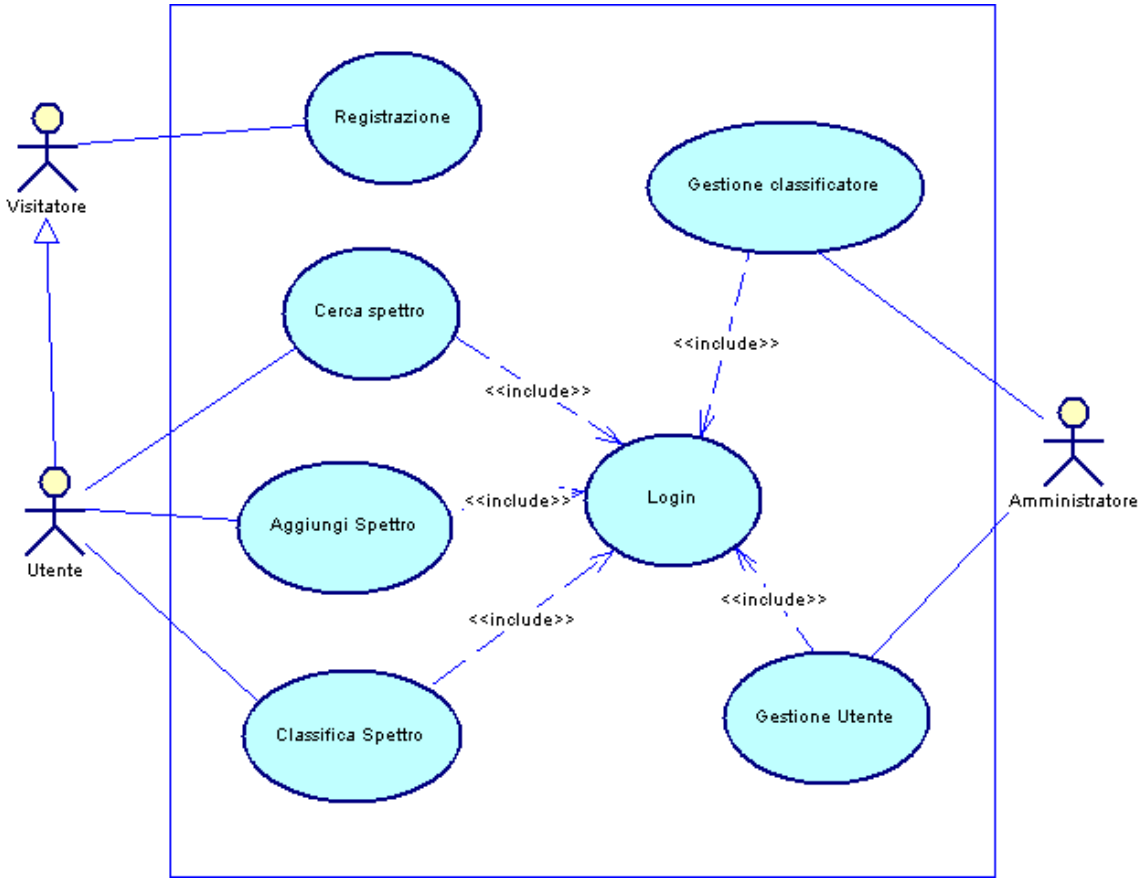
Estrae il features vector dalla peaklist sottoposta.

3.2.18.4 Output

Features vector.

Fine specifica dei requisiti

Par 5.3: Casi d'uso di Bacteria Classifier



Caso d'uso Registrazione

Scenario: …

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.

Caso d'uso Login

Scenario: login con successo.

1. L'utente accede al sistema.
2. Il sistema mostra il Form per l'accesso.
3. L'utente immette user name e password.
4. L'utente trasmette i dati.
5. Il sistema riceve i dati.
6. Il sistema verifica la corrispondenza in archivio dei dati immmessi.
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.

Caso d'uso Login

Scenario: login senza successo.

1. L'utente accede al sistema.
2. Il sistema mostra il Form per l'accesso.
3. L'utente immette user name e password.
4. L'utente trasmette i dati.
5. Il sistema riceve i dati.
6. Il sistema verifica la corrispondenza in archivio dei dati immmessi.
7. Il sistema non trova una corrispondenza in archivio.
8. Il sistema mostra un messaggio di accesso negato.
9. Il sistema ripropone il Form.
10. L'utente sceglie di riprovare o di rinunciare.

Caso d'uso Ricerca Spettro

Scenario: ricerca con esito positivo

1. <<include>> " **Login** " con successo.
2. L'utente chiede di cercare uno spettro.
3. Il sistema mostra un Form per immettere i dati sul batterio dello spettro.
4. L'utente compila il Form con dati specifici del batterio.
5. L'utente trasmette i dati.

6. Il sistema riceve i dati.
7. Il sistema cerca nell'archivio gli spettri relativi ai dati immessi dall'utente.
8. Il sistema trova uno o più spettri corrispondenti.
9. Il sistema visualizza il o gli spettri trovati corrispondenti al batterio.

Caso d'uso Ricerca Spettro

Scenario: ricerca con esito negativo

1. <<include>> " **Login** " con successo.
2. L'utente chiede di cercare uno spettro.
3. Il sistema mostra un Form per immettere i dati sul batterio dello spettro.
4. L'utente compila il Form con dati specifici del batterio.
5. L'utente trasmette i dati.
6. Il sistema riceve i dati.
7. Il sistema cerca nell'archivio gli spettri relativi ai dati immessi dall'utente.
8. Il sistema non trova spettri corrispondenti.
9. Il sistema visualizza un messaggio di ricerca senza risultati.
10. Il sistema visualizza nuovamente il Form per la ricerca di spettri.
11. L'utente sceglie di rinunciare o di ricercare un altro batterio.

Caso d'uso Aggiunta Spettro

Scenario: nomenclatura batterica valida

1. <<include>> " **Login** " con successo.
2. L'utente chiede di aggiungere un suo spettro all'archivio.
3. Il sistema visualizza un Form per l'immissione dello spettro e dei dati relativi.
4. L'utente compila il Form.
5. L'utente trasmette i dati e lo spettro.
6. Il sistema riceve i dati e lo spettro.
7. Il sistema verifica che i dati corrispondano ad una valida nomenclatura e che lo spettro sia valido.
8. Il sistema riporta dati validi.

9. Il sistema registra lo spettro.
10. Il sistema trasmette un messaggio di avvenuta archiviazione.

Caso d'uso Aggiunta Spettro

Scenario: nomenclatura batterica non valida

1. <<include>> " **Login** " con successo.
2. L'utente chiede di aggiungere un suo spettro all'archivio.
3. Il sistema visualizza un Form per l'immissione dello spettro e dei dati relativi.
4. L'utente compila il Form.
5. L'utente trasmette i dati e lo spettro.
6. Il sistema riceve i dati e lo spettro.
7. Il sistema verifica che i dati corrispondano ad una valida nomenclatura e che lo spettro sia valido.
8. Il sistema rileva una violazione nei dati sul batterio.
9. Il sistema visualizza un messaggio di nomenclatura batterica non valida.
10. Il sistema visualizza nuovamente il Form per l'immissione di spettro.
11. L'utente riprova o desiste.

Caso d'uso Classifica Spettro

Scenario: …

1. <<include>> " **Login** " con successo.
2. L'utente chiede di classificare uno spettro.
3. Il sistema mostra un Form per l'immissione dello spettro.
4. Il sistema verifica che il formato dello spettro è valido.
5. Il sistema mostra un Form per la scelta di parametri per l'eventuale l'pre-trattamento dello spettro.
6. L'utente sceglie i parametri.
7. Il sistema mostra l'anteprima degli effetti del trattamento dello spettro.
8. L'utente valida la scelta.
9. Il sistema mostra i classificatori disponibili.
10. L'utente sceglie un classificatore.
11. Il sistema effettua la classificazione.

12. Il sistema mostra i risultati.

Caso d'uso Gestione Classificatore

Scenario: classificatore aggiunto con successo

1. <<include>> " **Login** " con successo.
2. L'amministratore sceglie di aggiungere un classificatore.
3. Il sistema mostra un Form per l'aggiunta del classificatore con i dati relativi
4. L'amministratore compila il Form.
5. L'amministratore trasmette dati e classificatore.
6. Il sistema verifica il classificatore.
7. Il sistema mostra un messaggio di classificatore valido.
8. L' amministratore chiede di addestrare il classificatore.
9. Il sistema addestra il classificatore con il dataset disponibile.
10. Il sistema visualizza statistiche di classificazione.

Caso d'uso Gestione Classificatore

Scenario: addestra classificatore

1. <<include>> " **Login** " con successo.
2. L'amministratore sceglie di addestrare un classificatore.
3. Il sistema mostra un Form per l'addestramento del classificatore con i dataset e i classificatori disponibili.
4. L'amministratore compila il Form.
5. L'amministratore trasmette i dati.
6. Il sistema addestra il classificatore.
7. Il sistema mostra un messaggio con le statistiche dell'addestramento.

Caso d'uso Gestione Classificatore

Scenario: cancellazione classificatore

1. <<include>> " **Login** " con successo.

2. L'amministratore sceglie di cancellare un classificatore.
3. Il sistema mostra i classificatori disponibili e i relativi dataset con cui sono stati addestrati.
4. L'amministratore sceglie un classificatore.
5. Il sistema cancella il classificatore.
6. Il sistema mostra un messaggio di classificatore cancellato.

Caso d'uso Gestione Utente

Scenario: interdizione utente

1. <<include>> " **Login** " con successo.
2. L'amministratore sceglie di interdire un utente.
3. Il sistema mostra gli utenti iscritti.
4. L'amministratore sceglie un utente da interdire.
5. Il sistema pone un utente in stato interdetto.
6. Il sistema mostra un messaggio con lo stato dell'utente.

Caso d'uso Gestione Utente

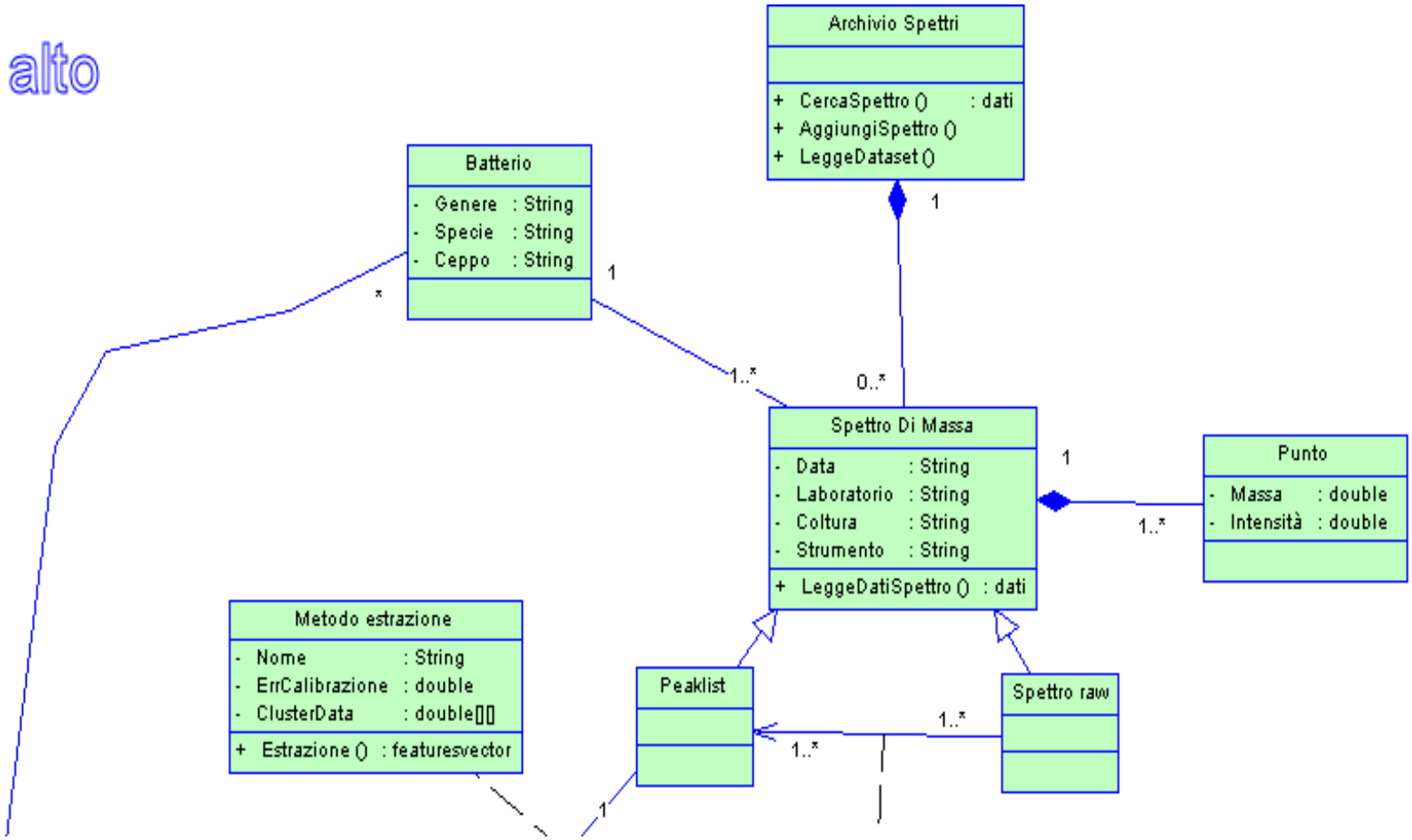
Scenario: revoca interdizione utente

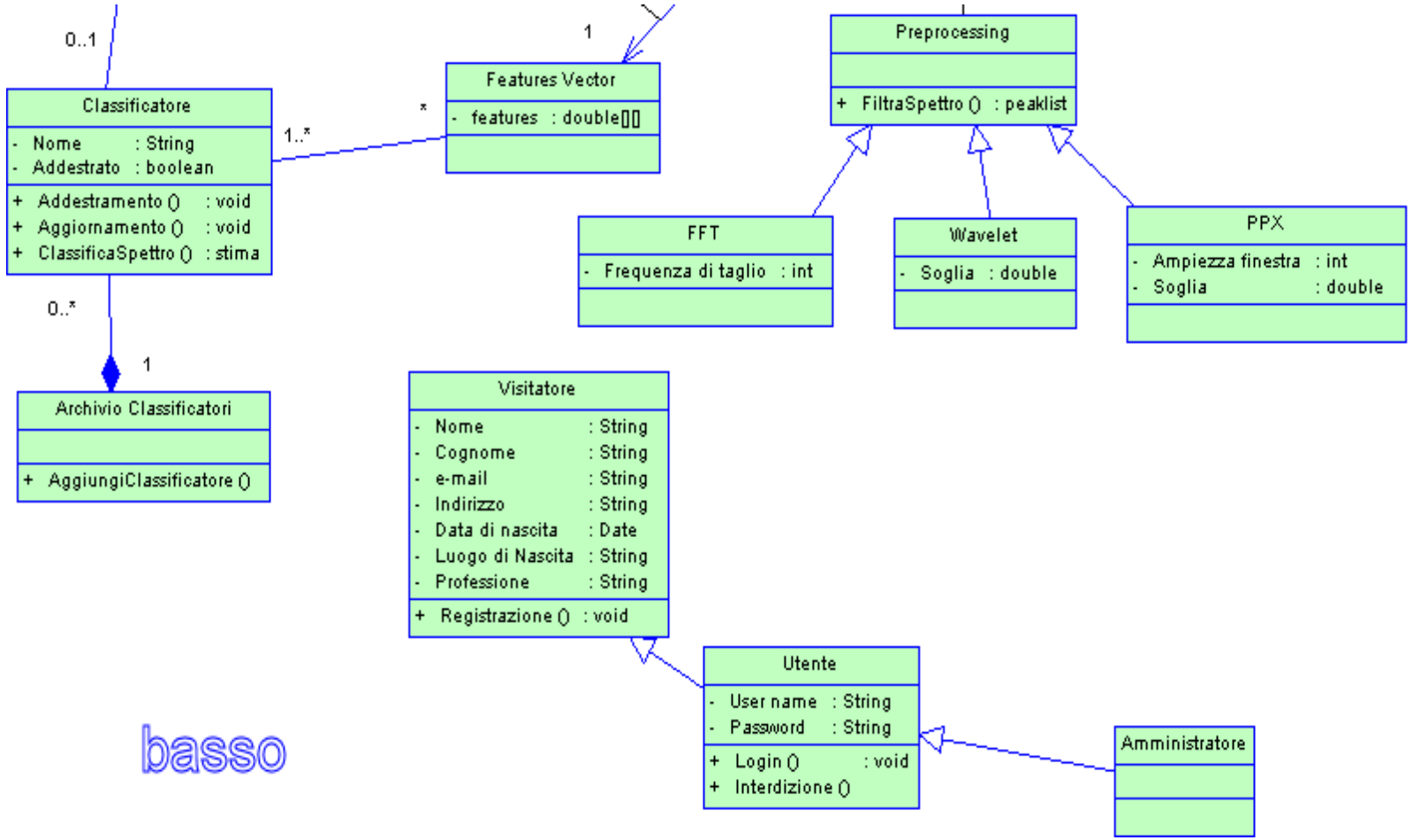
1. <<include>> " **Login** " con successo.
2. L'amministratore sceglie di revocare un'interdizione per un utente.
3. Il sistema mostra gli utenti interdetti.
4. L'amministratore sceglie l'utente per la revoca dell'interdizione.
5. Il sistema pone un utente in stato non interdetto.
6. Il sistema mostra un messaggio con lo stato dell'utente.

Par 5.4: Class diagram livello concettuale

Il class diagram del livello concettuale rende una visione statica del dominio applicativo, con tutti i concetti di interesse per l'applicazione e tutti i legami fra loro. Esso rappresenta l'architettura ad alto livello dell'applicazione

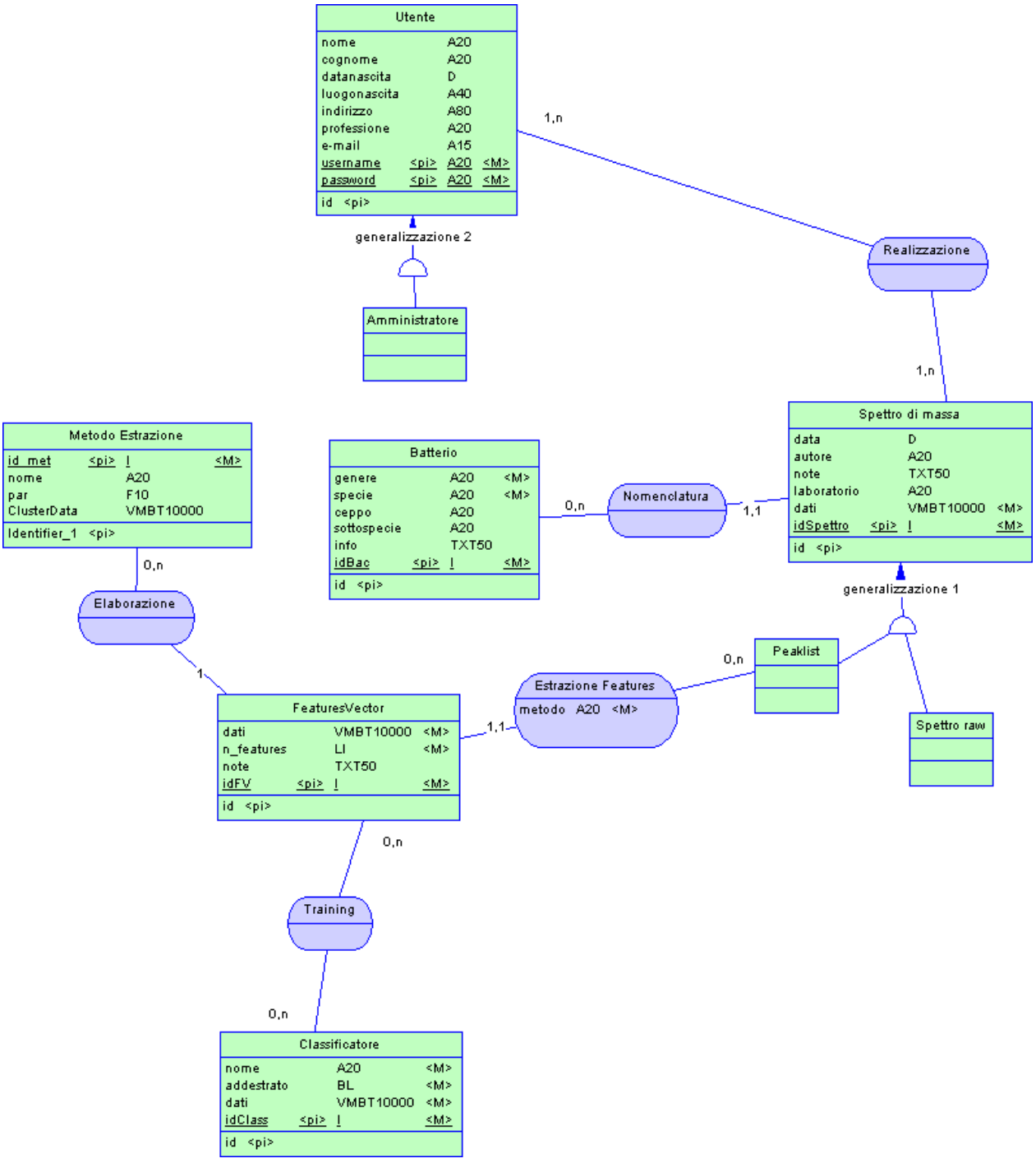
alto


Archivio Spettri
+ CercaSpettro () : dati
+ AggiungiSpettro ()
+ LeggeDataset ()
Batterio
- Genere : String
- Specie : String
- Ceppo : String
Spettro Di Massa
- Data : String
- Laboratorio : String
- Coltura : String
- Strumento : String
+ LeggeDatiSpettro () : dati
Punto
- Massa : double
- Intensità : double
Metodo estrazione
- Nome : String
- ErrCalibrazione : double
- ClusterData : double[][]
+ Estrazione () : featuresvector
Peaklist
Spettro raw
1
*
1..*
0..*
1
1..*
1..*
1..*
1

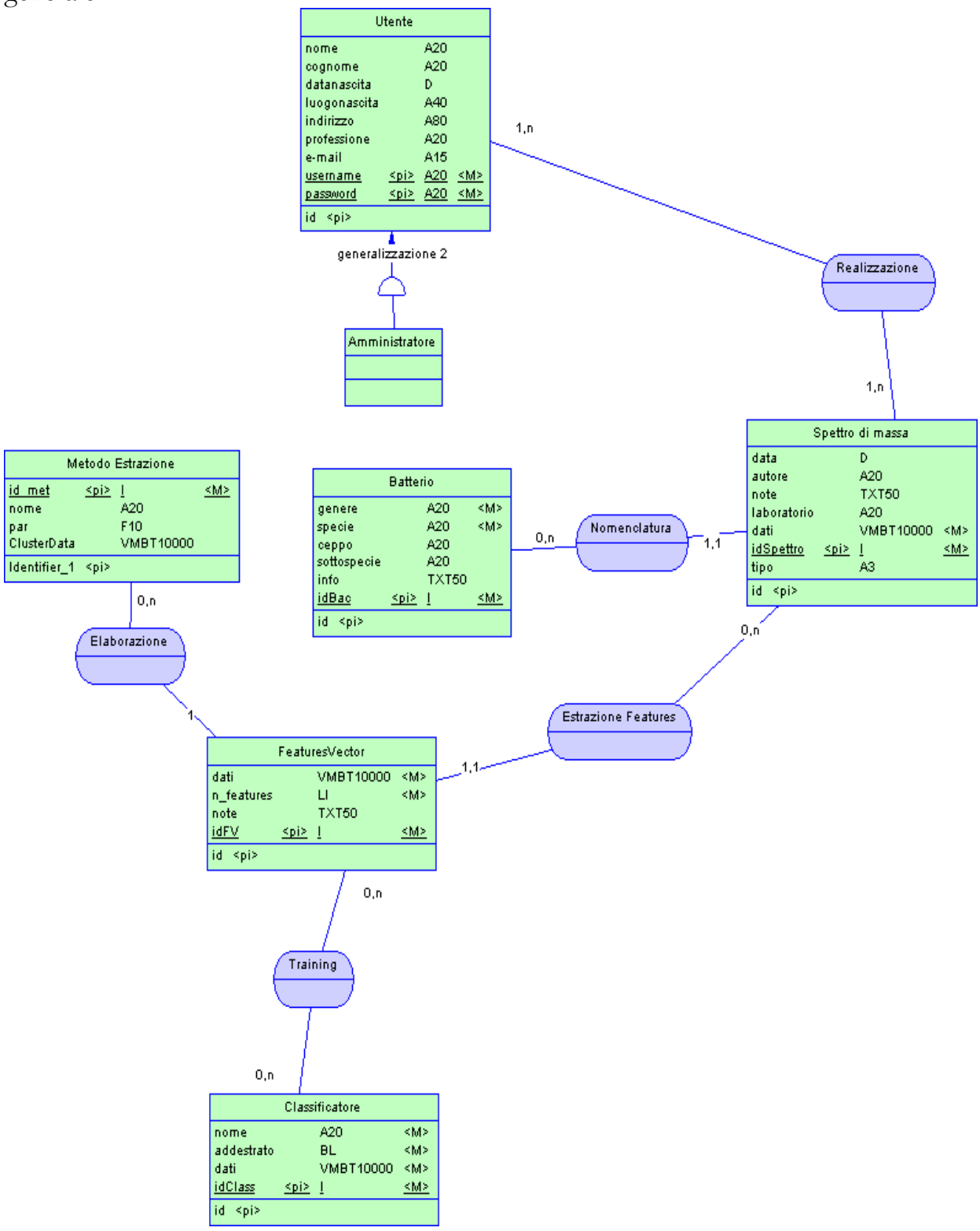

0..1
1
Classificatore
- Nome : String
- Addestrato : boolean
+ Addestramento () : void
+ Aggiornamento () : void
+ ClassificaSpettro () : stima
1..*
*
Features Vector
- features : double[][]
Preprocessing
+ FiltraSpettro () : peaklist
FFT
- Frequenza di taglio : int
Wavelet
- Soglia : double
PPX
- Ampiezza finestra : int
- Soglia : double
0..*
1
Archivio Classificatori
+ AggiungiClassificatore ()
Visitatore
- Nome : String
- Cognome : String
- e-mail : String
- Indirizzo : String
- Data di nascita : Date
- Luogo di Nascita : String
- Professione : String
+ Registrazione () : void
Utente
- User name : String
- Password : String
+ Login () : void
+ Interdizione ()
Amministratore
basso

Par 5.5: Progetto del database

Dai requisiti e dal class diagram di alto livello si possono trarre informazioni sui dati permanenti e quindi si può ideare il database. Alcune scelte sul database sono già state prese in fase di analisi e nella descrizione dei requisiti.
Lo schema concettuale E-R del database è il seguente:



La ristrutturazione dello schema ha comportato la sola esclusione della gerarchia che riguarda l'entità spettro. Si è scelto di accorpare le entità figlie nell'entità più generale:



Lo schema logico derivante dallo schema ristrutturato si può riassumere così:

bacteria(id, genus, species, strain, subspecies, info)

classifiers(id, name, class, trained)

featuresvectors(id, vector, n_features, notes, id_sp, id_method)

femethods(id, method, par, clusterdata)

spectra(id, data, type, idbacterium, author, notes)

trainings(id, id_cl, id_feat)

utenti(id,nome,cognome, indirizzo, datanascita, luogonascita, e_mail, username, passsword, tipo)

Sussistono i seguenti vincoli di integrità referenziale:

featuresvectors.id_sp → spectra.id
featuresvectors.id_method → femethods.id
trainings.id_cl → classifier.id
trainings.id_feat → featuresvectors.id

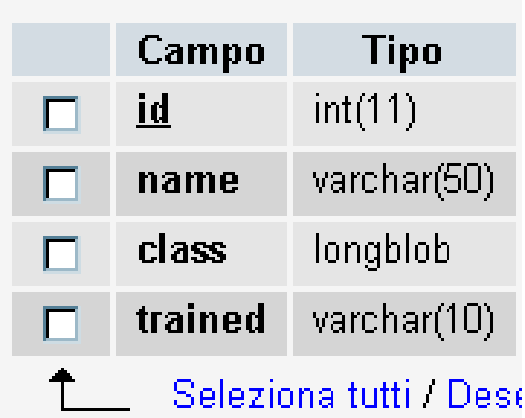
Inoltre, come regola aziendale, un classificatore non può essere addestrato con features vectors ricavati con metodi e parametri diversi; quindi sussiste un ulteriore vincolo legato all'addestramento di un classificatore che comporta che gli identificatori dei metodi di estrazione dei features vectors utilizzati per l'addestramento non differiscano da features vector a features vector.
L'implementazione del database è stata realizzata con phpadmin(un'interfaccia grafica, realizzata con il PHP, utile nella gestione di database MySQL). Le tabelle implementate con il phpadmin e con i domini degli attributi:

Bacteria

	Campo	Tipo
☐	**id**	int(10)
☐	**genus**	varchar(45)
☐	**species**	varchar(45)
☐	**strain**	varchar(45)
☐	**subspecies**	varchar(45)
☐	**info**	text

Seleziona tutti / Deselezi

Classifier



	Campo	Tipo
☐	**id**	int(11)
☐	**name**	varchar(50)
☐	**class**	longblob
☐	**trained**	varchar(10)

Seleziona tutti / Dese

Featuresvectors

	Campo	Tipo
☐	id	int(10)
☐	vector	blob
☐	n_features	int(20)
☐	notes	text
☐	id_sp	int(10)
☐	id_method	int(11)

Seleziona tutti / Dese

Femethods
(features extraction methods)

	Campo	Tipo
☐	id	int(11)
☐	method	varchar(50)
☐	par	double
☐	clustersdata	longblob

Seleziona tutti / Deselezior

Spectra

	Campo	Tipo
☐	id	int(10)
☐	data	blob
☐	type	varchar(3)
☐	idbacterium	int(10)
☐	author	varchar(45)
☐	notes	varchar(45)

Seleziona tutti / Deselezior

Trainings

	Campo	Tipo
☐	id_cl	int(11)
☐	id_feat	int(11)

Seleziona tutti /

Utenti

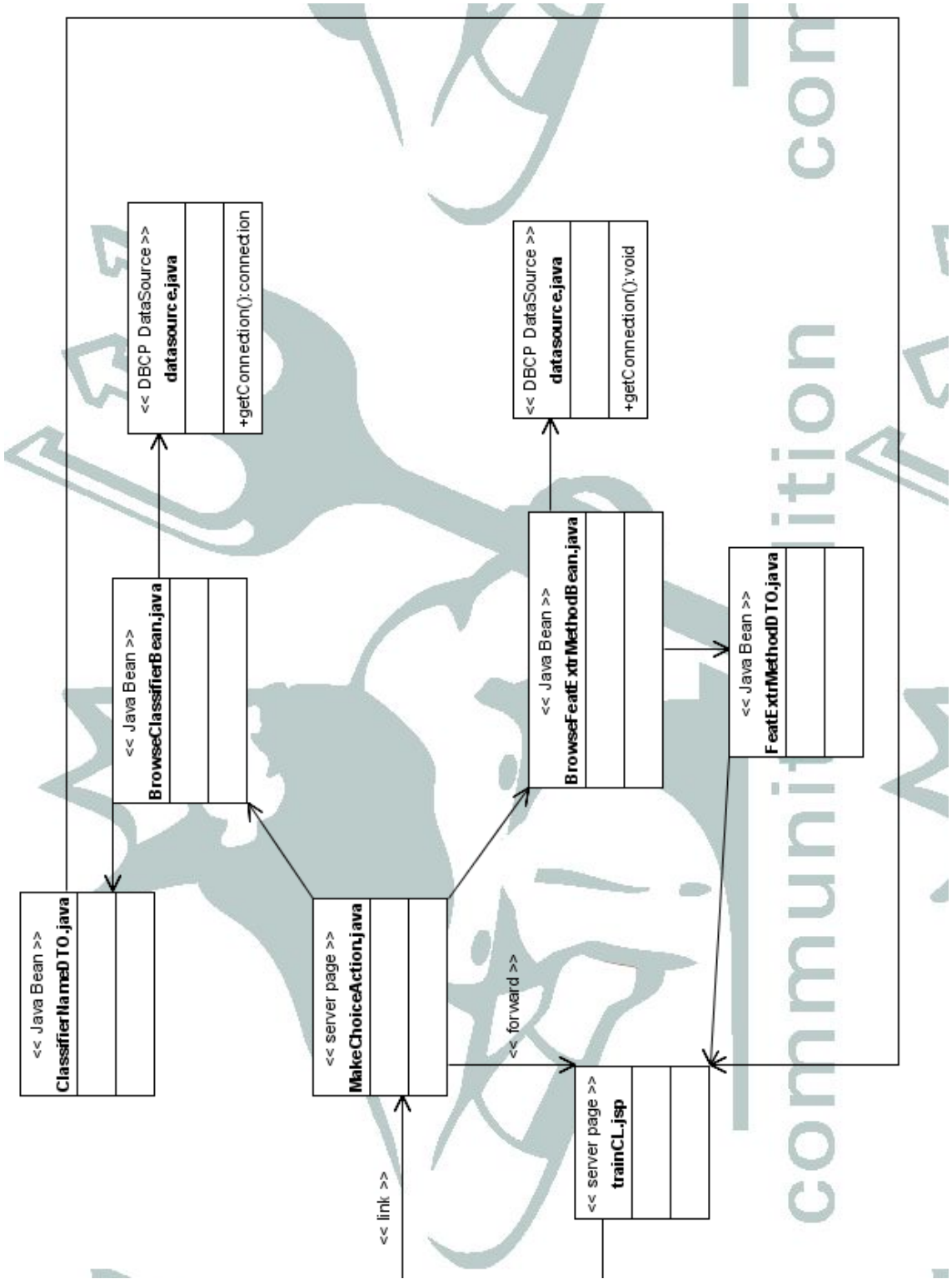
	Campo	Tipo
☐	**id**	int(11)
☐	**nome**	varchar(20)
☐	**cognome**	varchar(20)
☐	**indirizzo**	varchar(40)
☐	**datanascita**	date
☐	**luogonascita**	varchar(40)
☐	**e_mail**	varchar(10)
☐	**username**	varchar(10)
☐	**password**	varchar(10)
☐	**tipo**	varchar(3)


Seleziona tutti / Deselezion

Par 5.6: Class Diagrams livello progetto

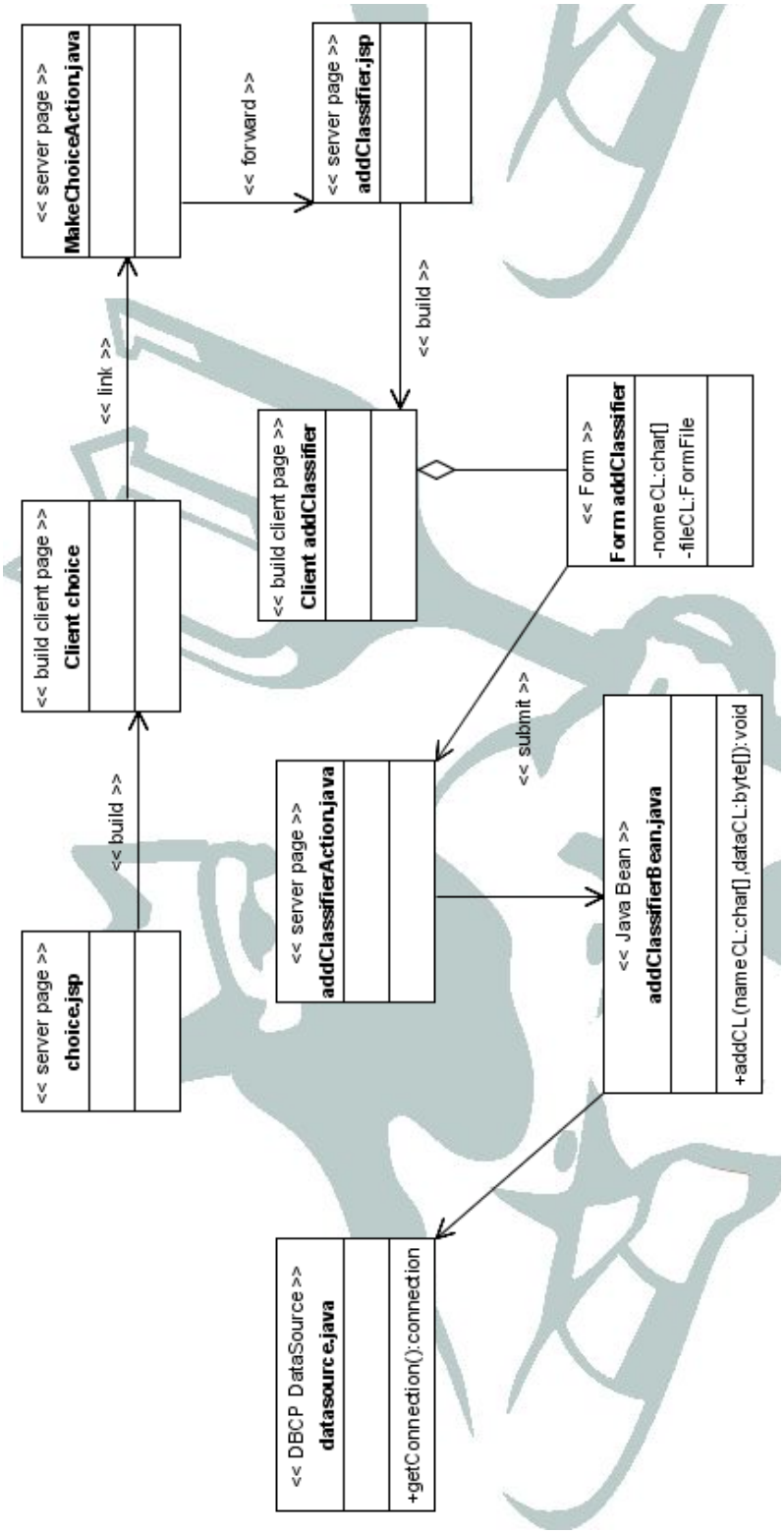
Seguono i class diagrams del progetto, che rappresentano la vista statica dell'applicazione, evidenziando con maggiore dettaglio l'architettura del sistema. Ogni class diagram è realativo ai casi d'uso con i possibili scenari. I diagrammi sono disposti in ordine alfabetico. Alcuni di essi sono divisi in due per chiarezza.

Class Diagram Design Level
Addestra Classificatore

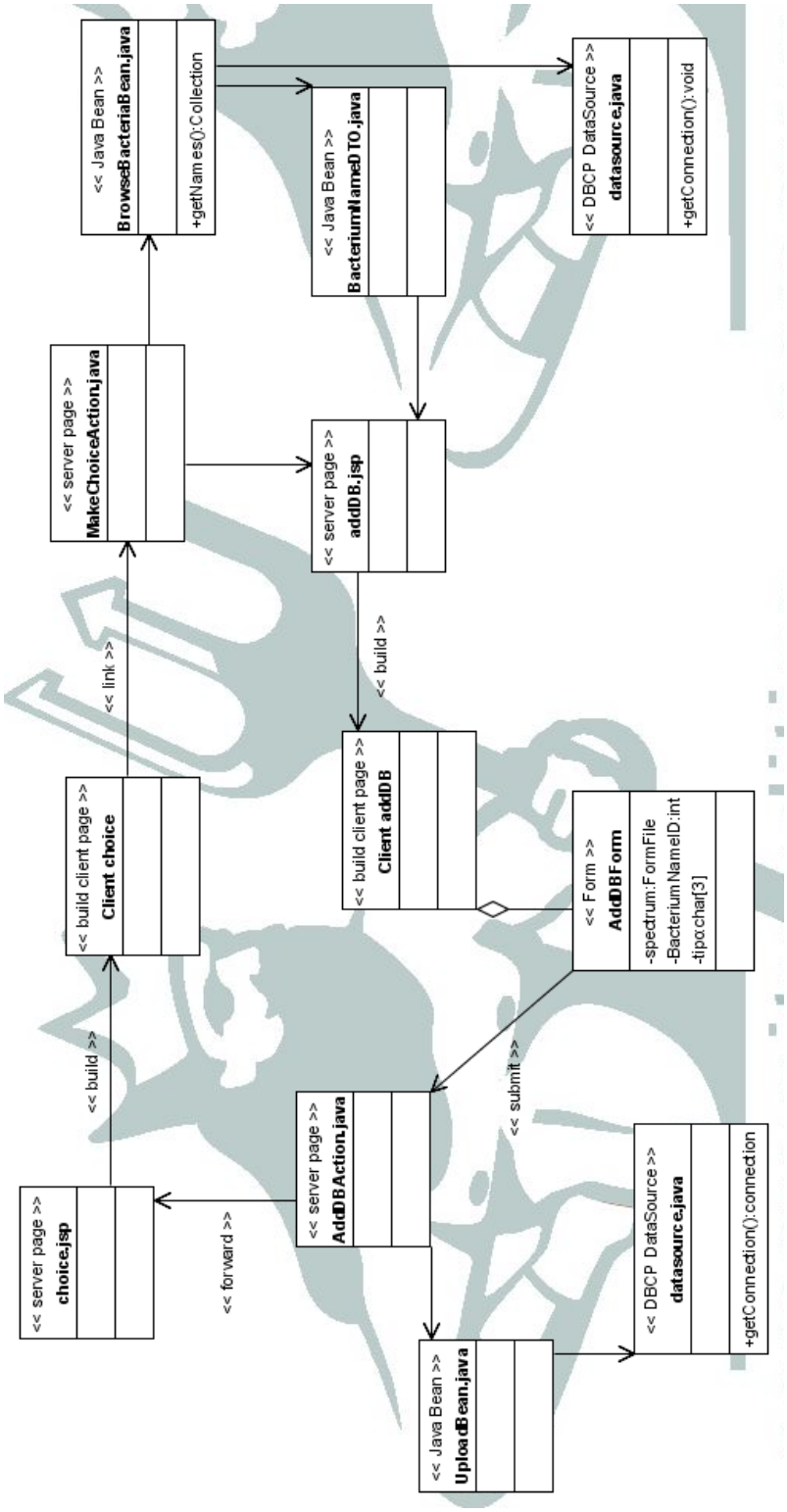


<< build client page >>
Client choice
<< build >>
<< server page >>
choice.jsp
<< build client page >>
Client trainCL
<< build >>
<< Form >>
Formtrainer
-id_cl:int
-id_feMeth:int
<< submit >>
<< server page >>
trainerAction.java
<< build client page >>
Client trainResults
<< build >>
<< server page >>
trainResults.jsp
<< forward >>
<< Java Bean >>
TrainerBean.java
+train(id_cl:int,idfeMeth:int):String
<< DBCP DataSource >>
datasource.java
+getConnection():void

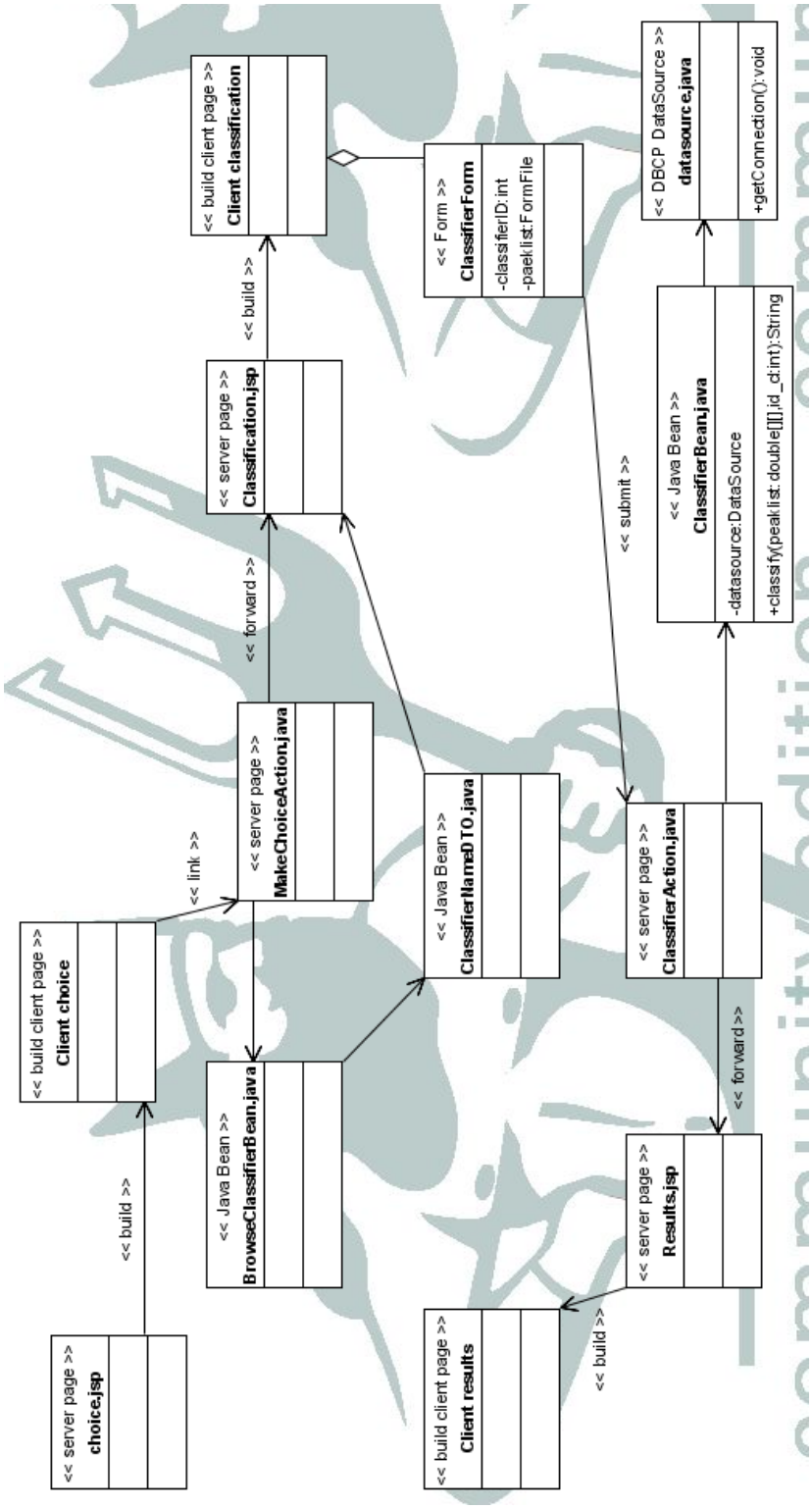
Class Diagram Design Level
Aggiunta Classificatore



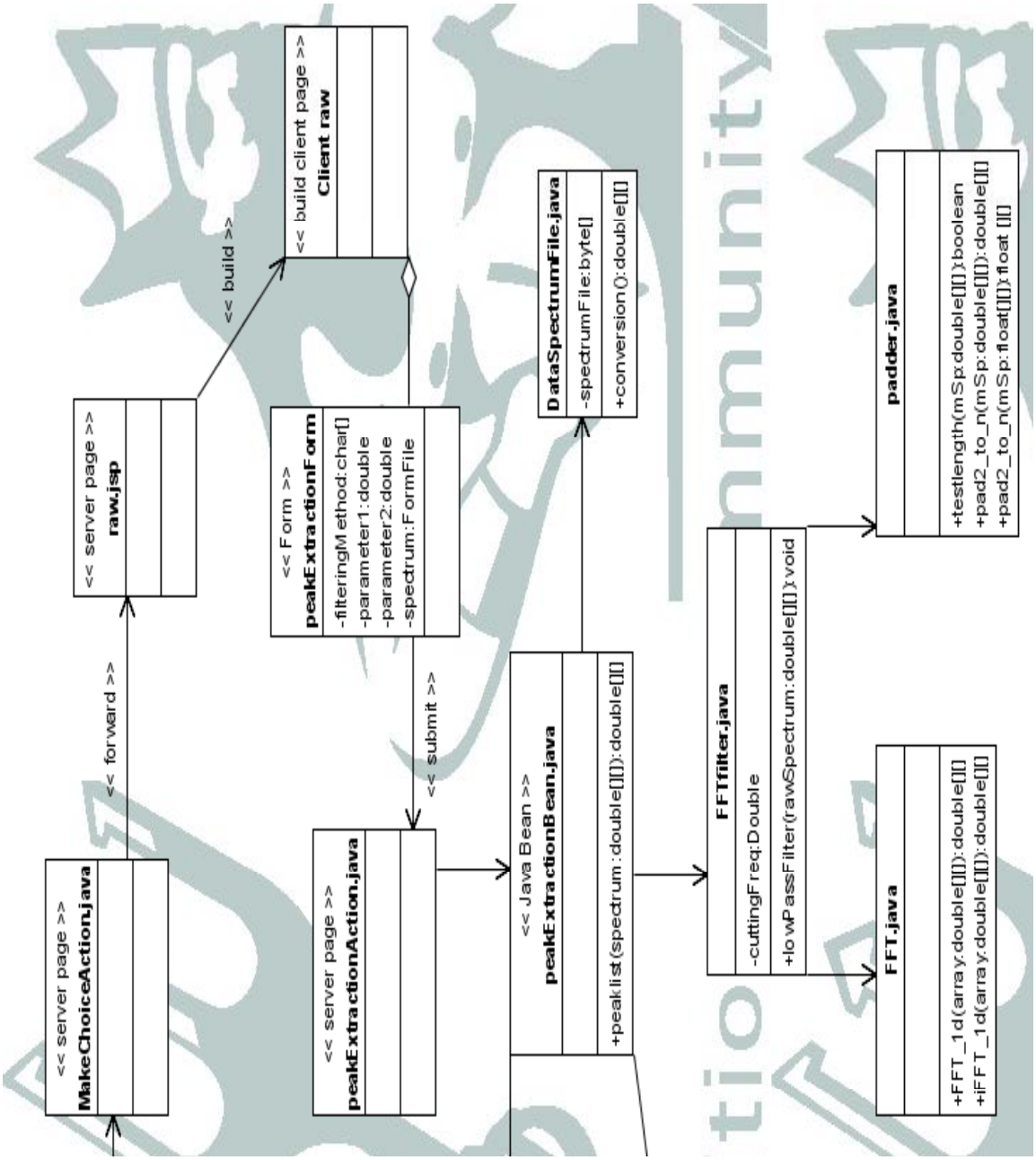
Class Diagram Design Level
Aggiunta Spettro



Class Diagram Design Level
Classifica peaklist

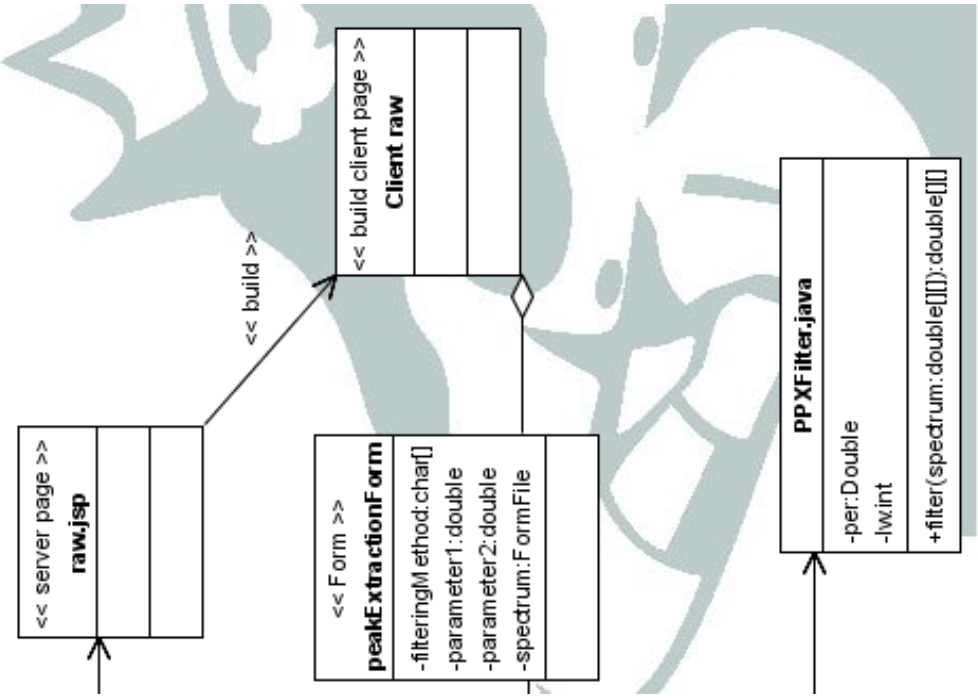


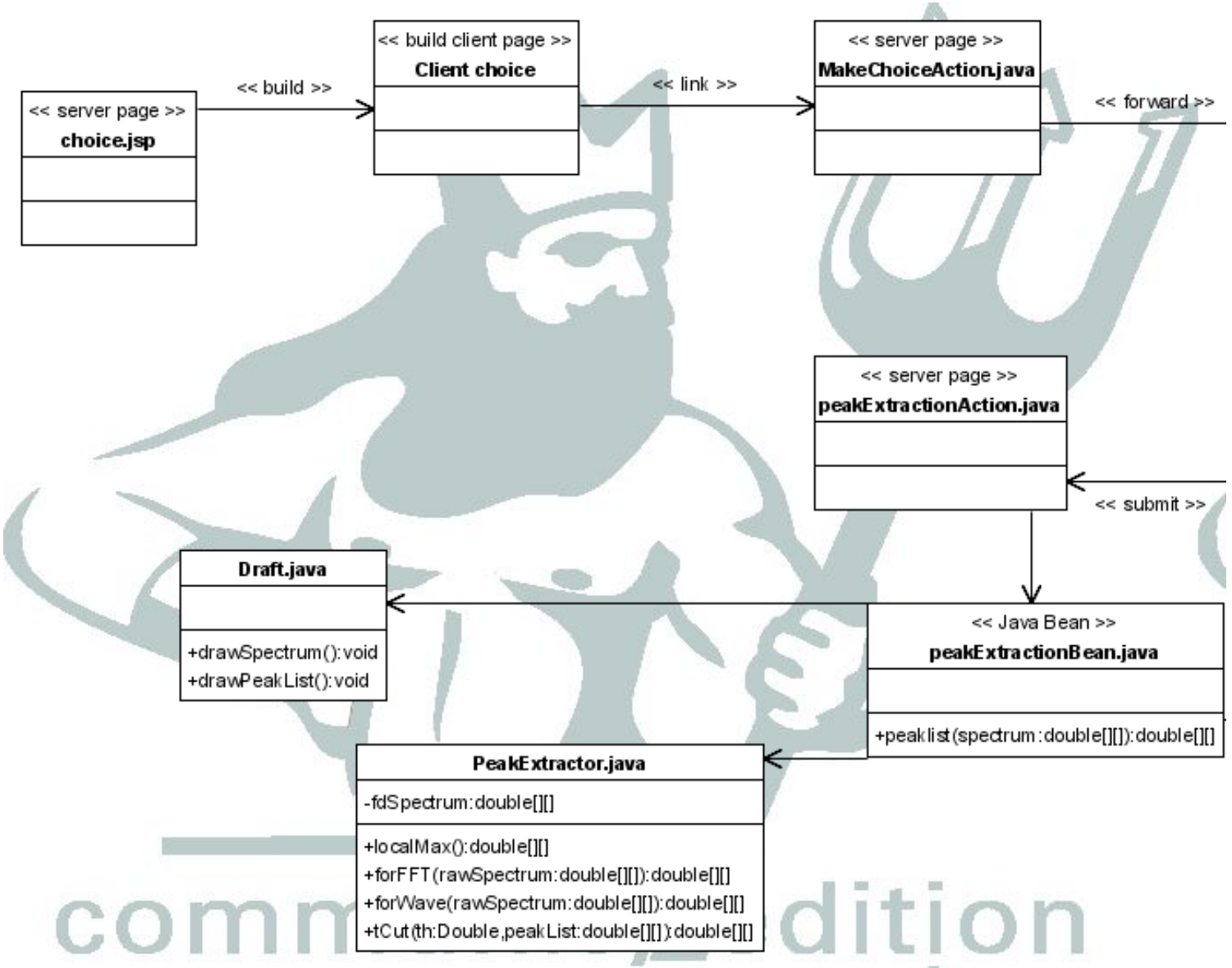
Class Diagram Design Level
Filtraggio Spettro Grezzo con FFT



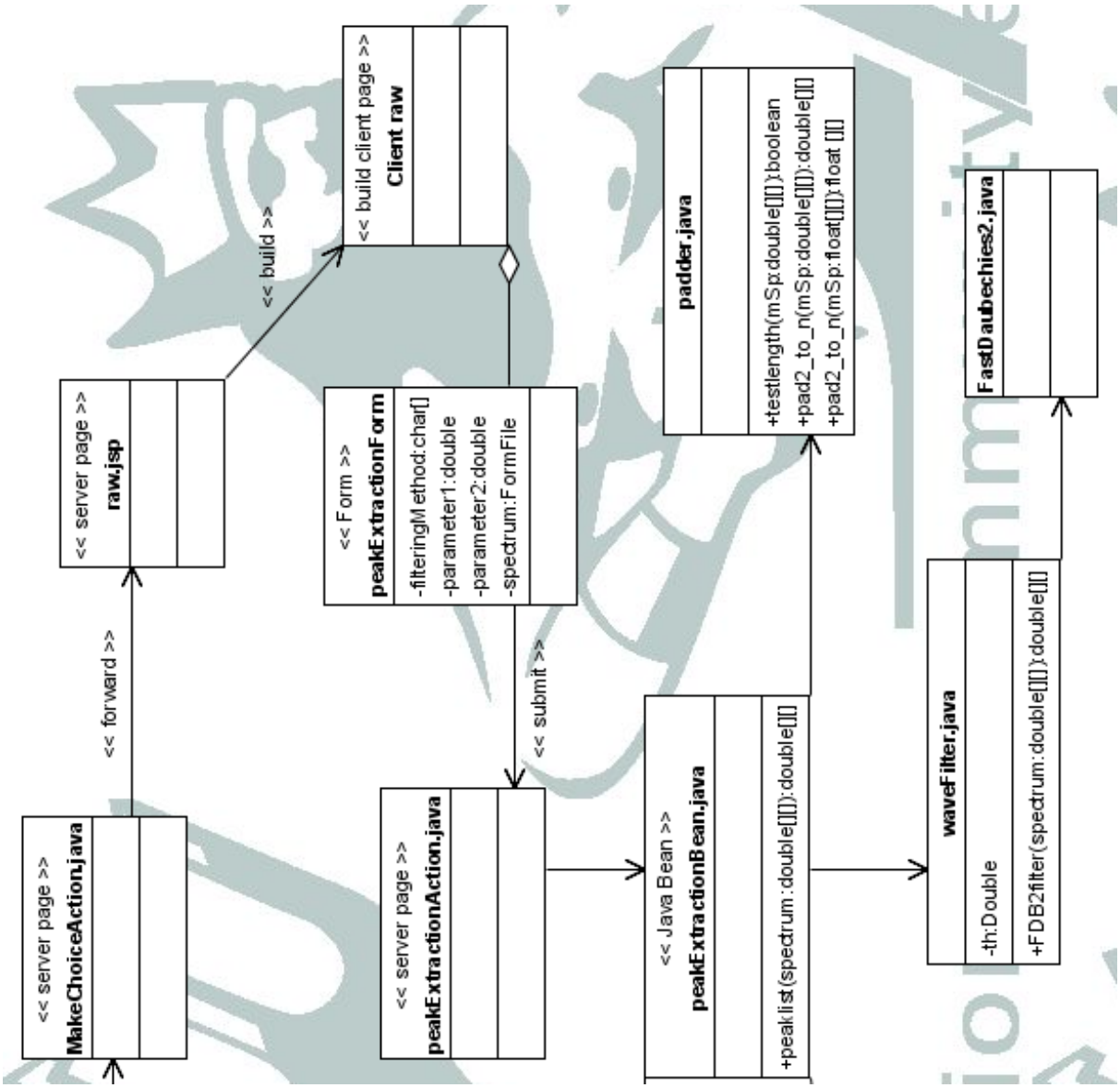
<< server page >>
choice.jsp
<< build >>
<< build client page >>
Client choice
<< link >>
Draft.java
+drawSpectrum():void
+drawPeakList():void
PeakExtractor.java
-fdSpectrum:double[][]
+localMax():double[][]
+forFFT(rawSpectrum:double[][]):double[][]
+forWave(rawSpectrum:double[][]):double[][]
+tCut(th:Double,peakList:double[][]):double[][]

Class Diagram Design Level
Filtraggio Spettro Grezzo con PPX



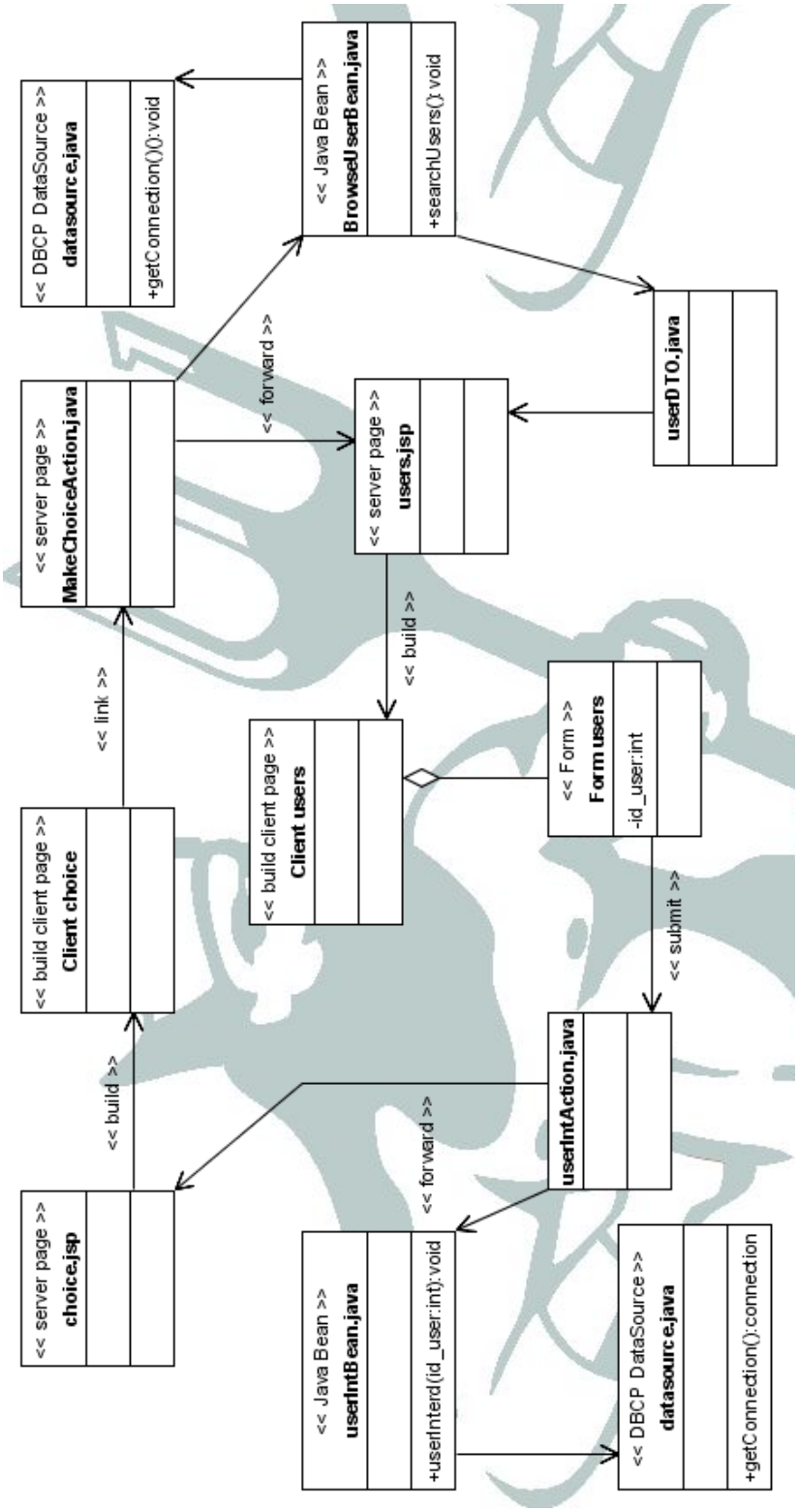

<< server page >>
choice.jsp
<< build >>
<< build client page >>
Client choice
<< link >>
<< server page >>
MakeChoiceAction.java
<< forward >>
<< server page >>
peakExtractionAction.java
<< submit >>
Draft.java
+drawSpectrum():void
+drawPeakList():void
<< Java Bean >>
peakExtractionBean.java
+peaklist(spectrum:double[][]):double[][]
PeakExtractor.java
-fdSpectrum:double[][]
+localMax():double[][]
+forFFT(rawSpectrum:double[][]):double[][]
+forWave(rawSpectrum:double[][]):double[][]
+tCut(th:Double,peakList:double[][]):double[][]

Class Diagram Design Level
Filtraggio Spettro Grezzo con Wavelet

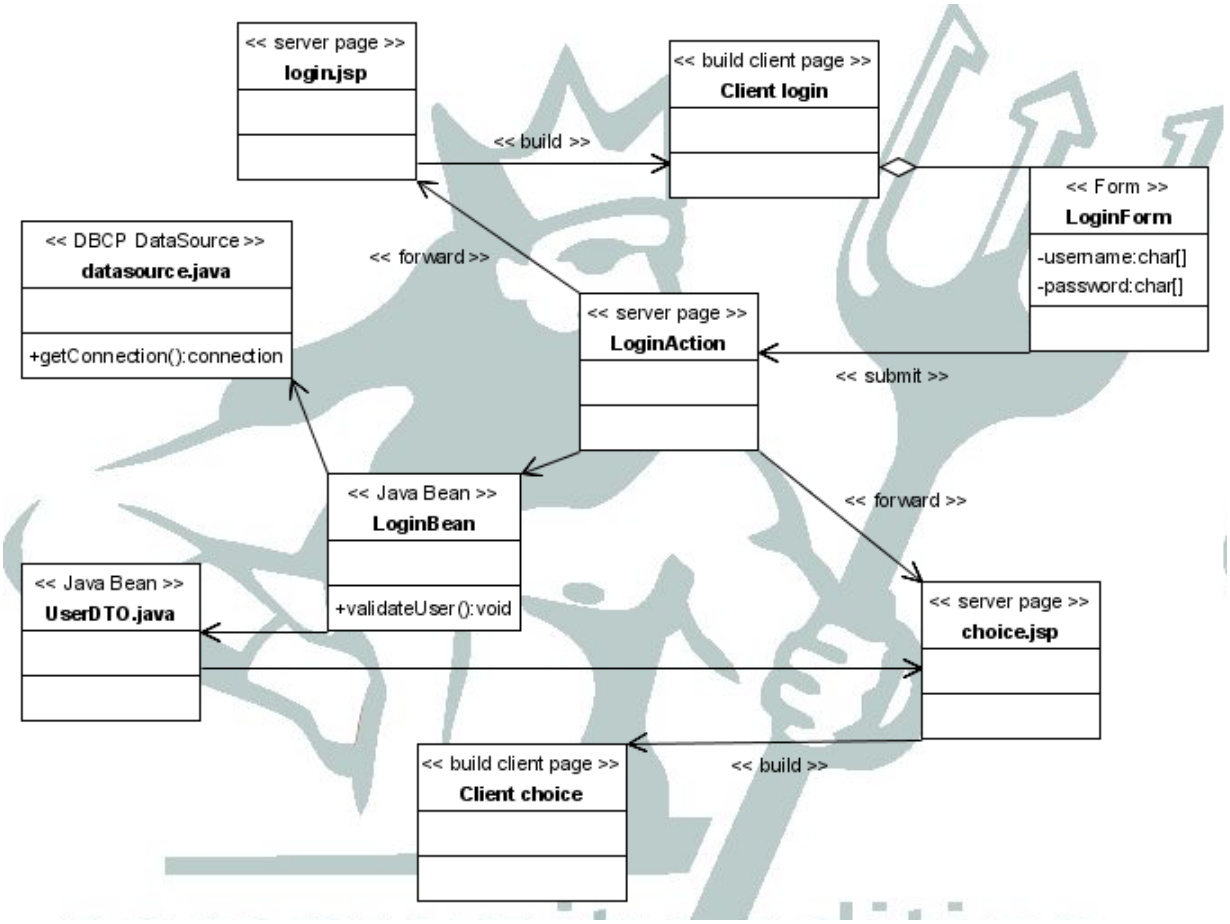


<< server page >>
choice.jsp
<< build >>
<< build client page >>
Client choice
<< link >>
Draft.java
+drawSpectrum():void
+drawPeakList():void
PeakExtractor.java
-fdSpectrum:double[][]
+localMax():double[][]
+forFFT(rawSpectrum:double[][]):double[][]
+forWave(rawSpectrum:double[][]):double[][]
+tCut(th:Double,peakList:double[][]):double[][]

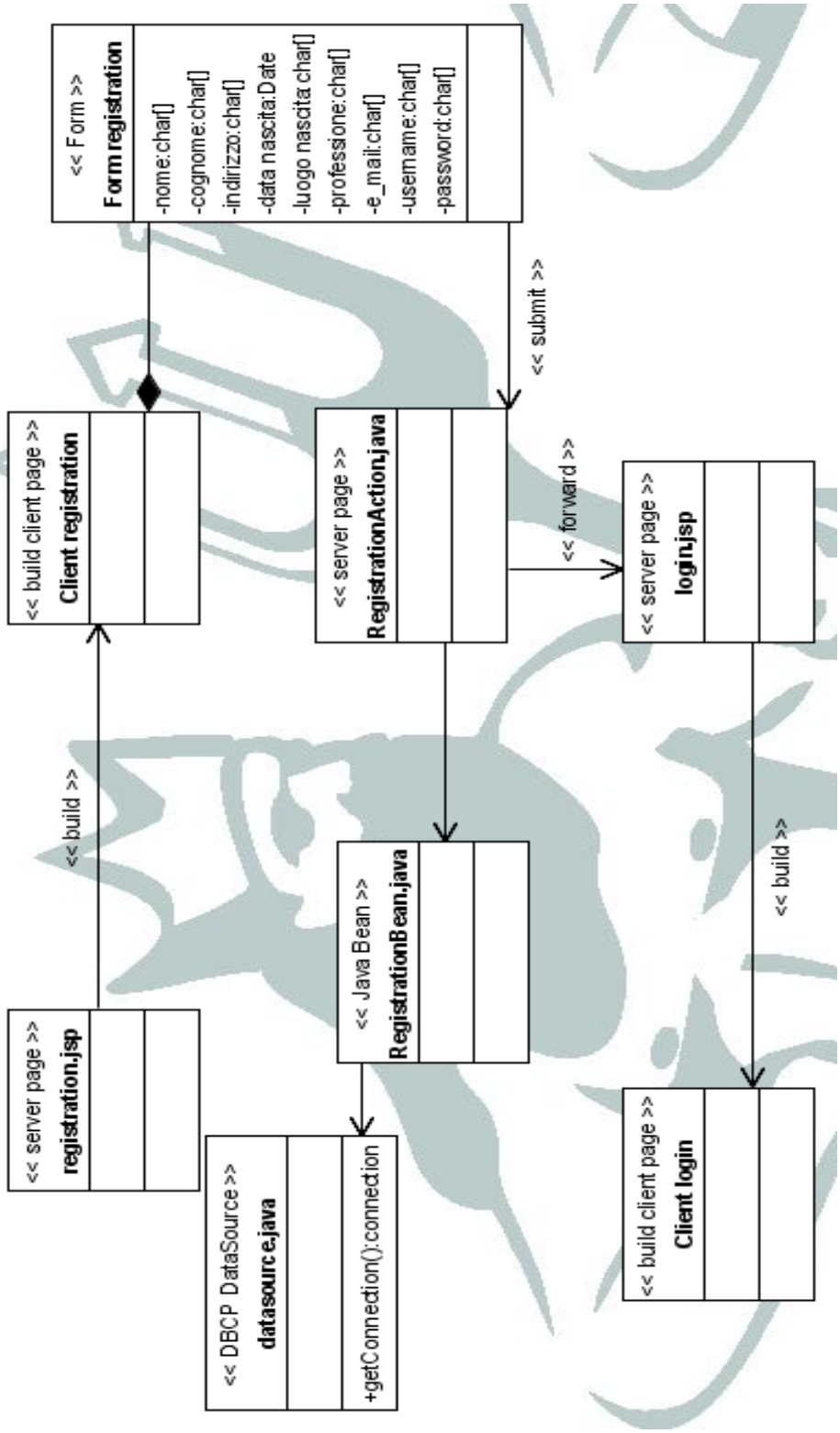
Class Diagram Design Level
Interdizione Utente



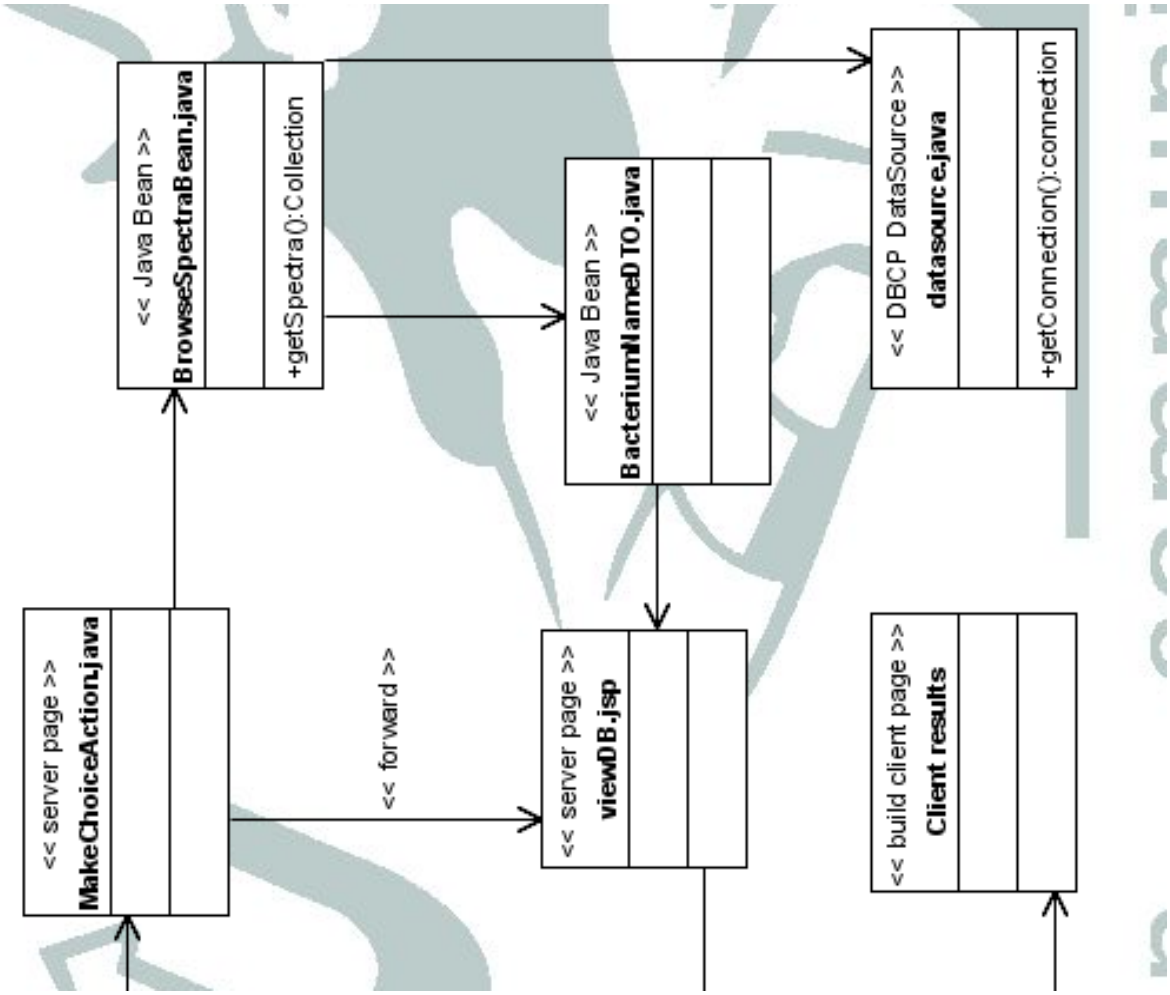
Class Diagram Design Level
Login



Class Diagram Design Level
Registrazione



Class Diagram Design Level
Ricerca Spettro

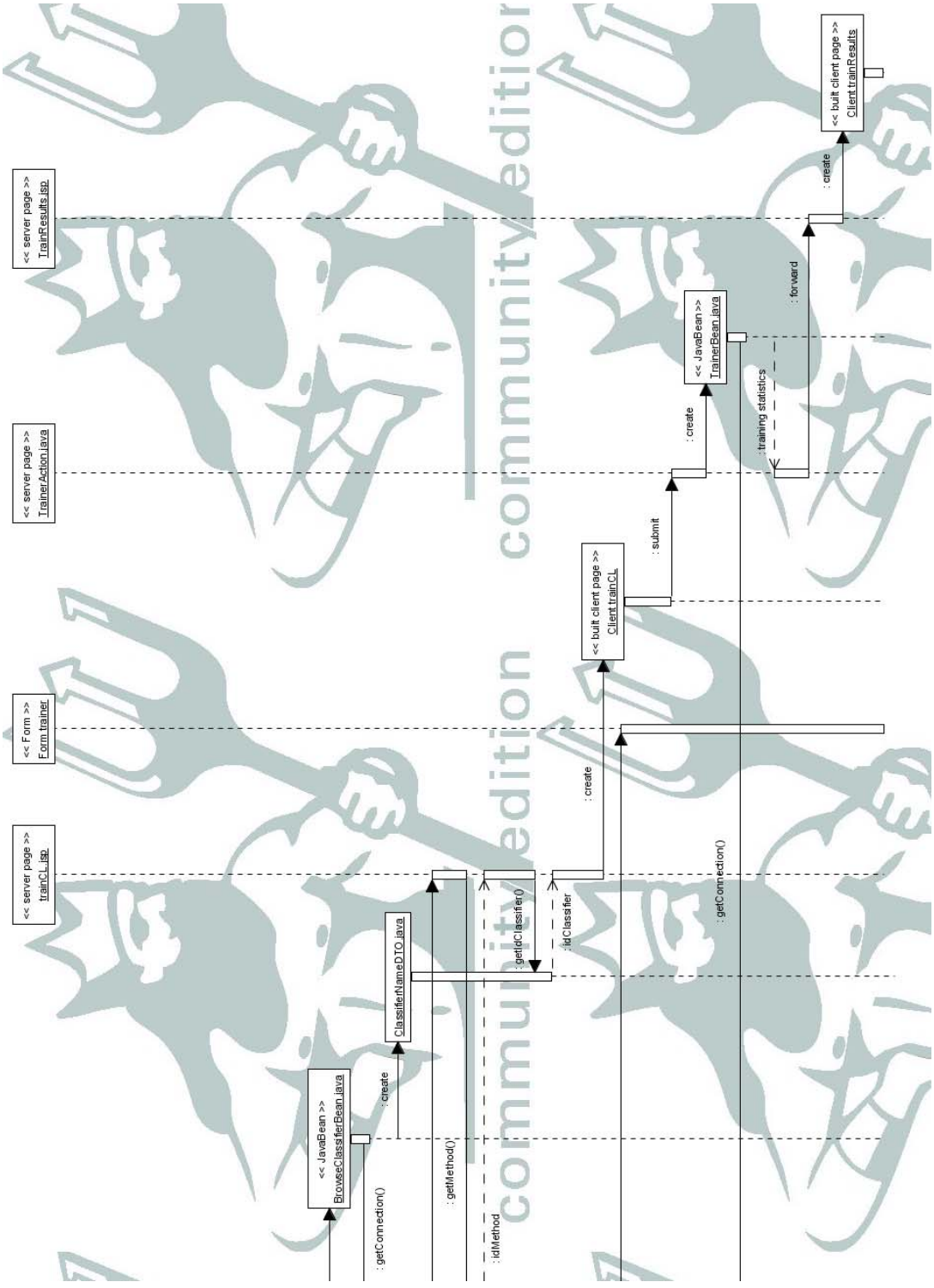


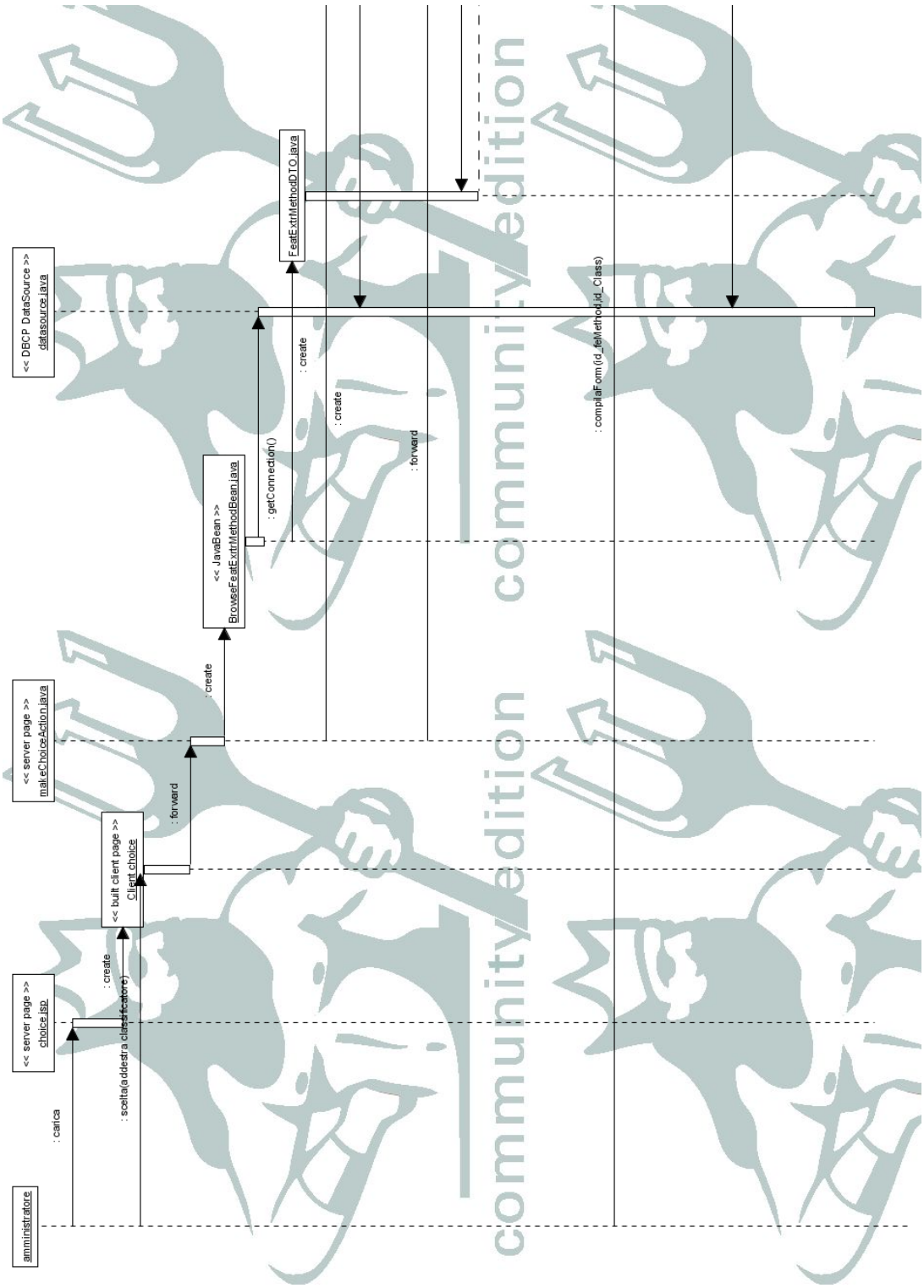
<< server page >>
choice.jsp
<< build >>
<< build client page >>
Client choice
<< link >>
<< Form >>
ViewForm
-idSpectrum:int
<< submit >>
<< build client page >>
Client viewDB
<< server page >>
viewDBAction.java
<< server page >>
results.jsp
<< build >>
<< DBCP DataSource >>
datasource.java
+getConnection():void
<< Java Bean >>
ViewSpectrumBean.java
+draw():void
Draft.java
+drawSpectrum():void
+drawPeakList():void

Par 5.7: Sequence Diagrams livello progetto

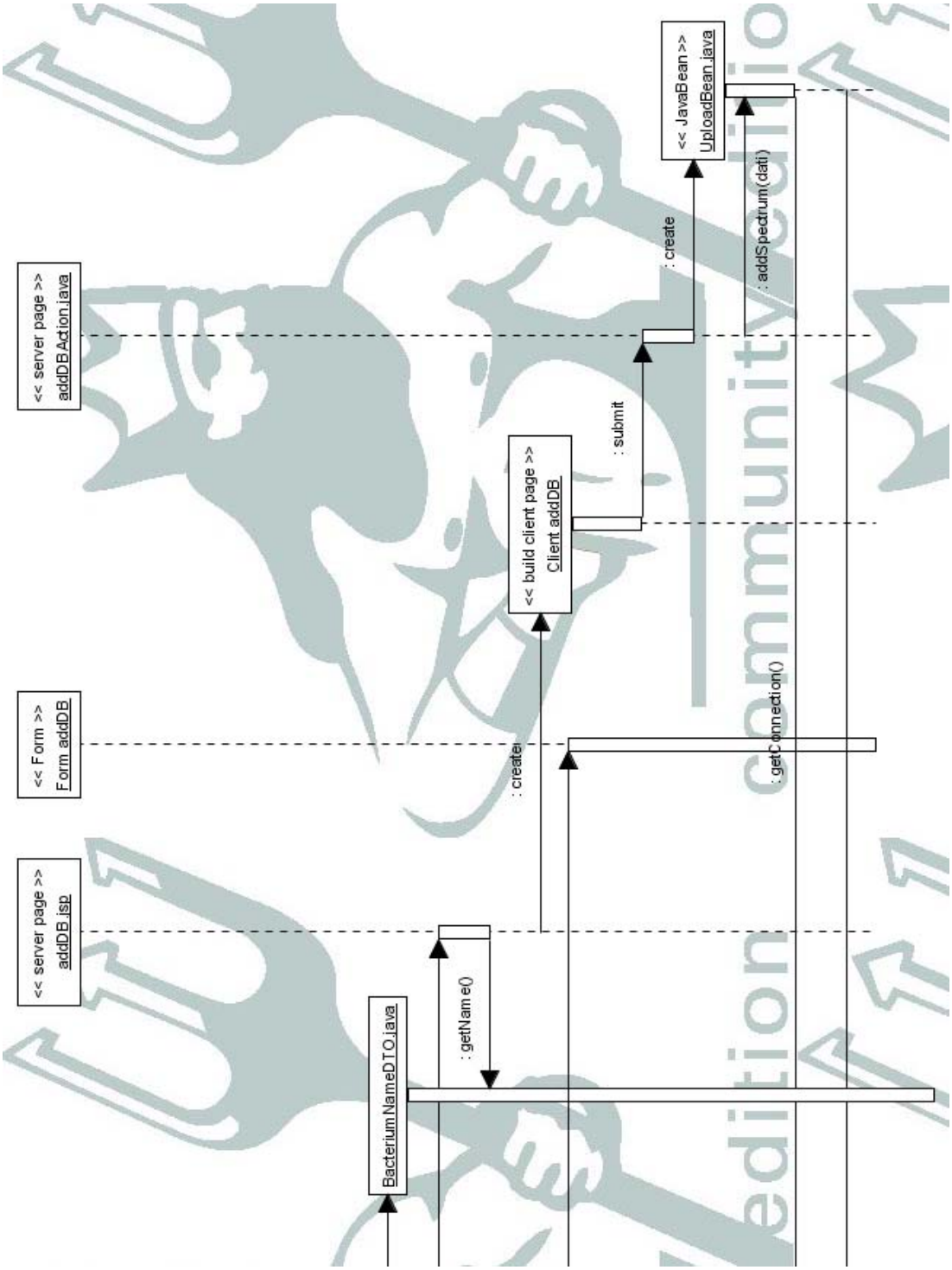
I sequence diagrams del livello progetto forniscono una visione dinamica del sistema, evidenziando le interazioni tra gli oggetti, ed in particolare come gli oggetti collaborano per realizzare le funzionalità dell'applicazione.
Anche, in questo caso, alcuni diagrammi sono stati divisi in due per chiarezza.

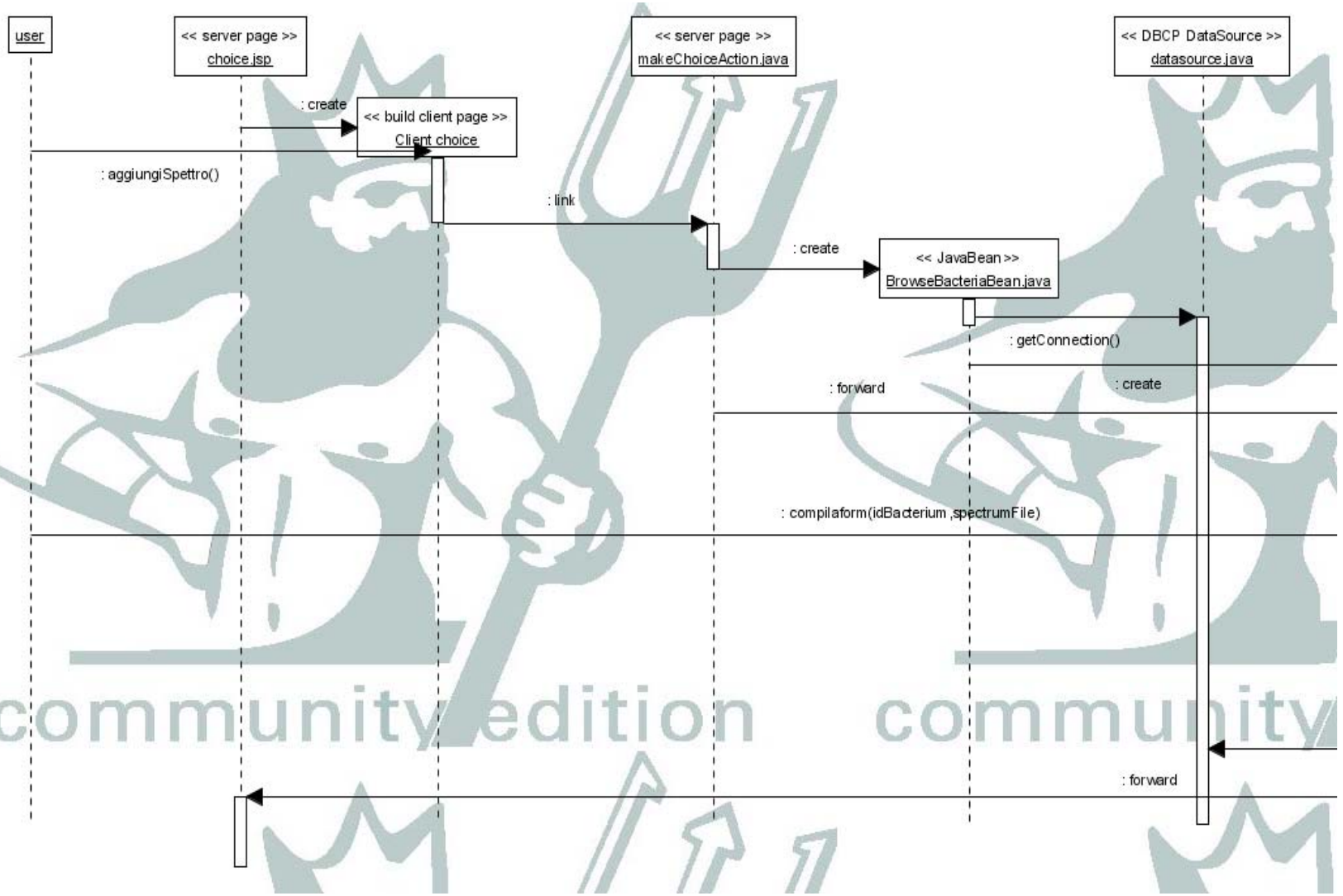
Sequence Diagram Design Level
Addestra Classificatore



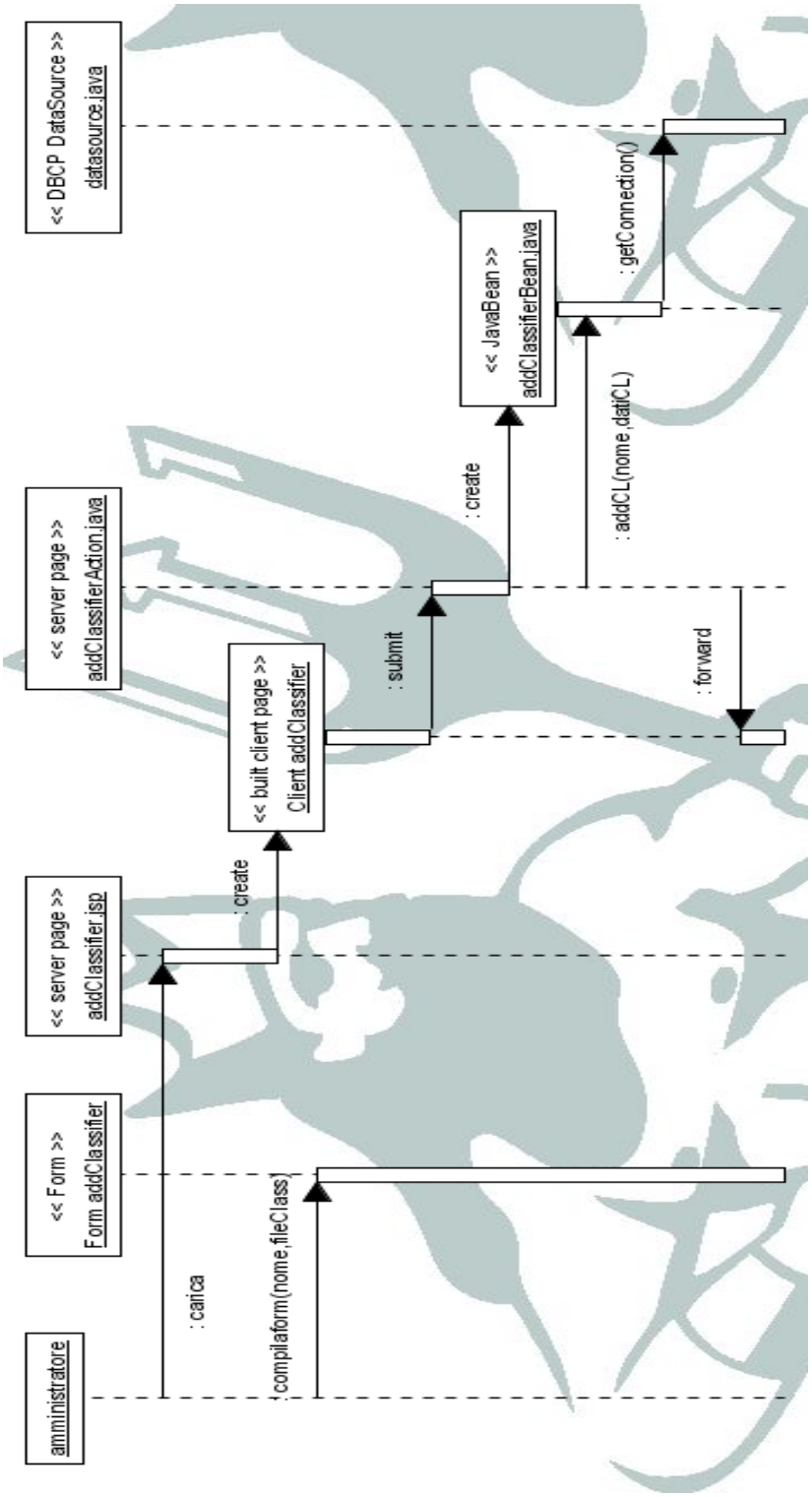


Sequence Diagram Design Level
Aggiungi Spettro

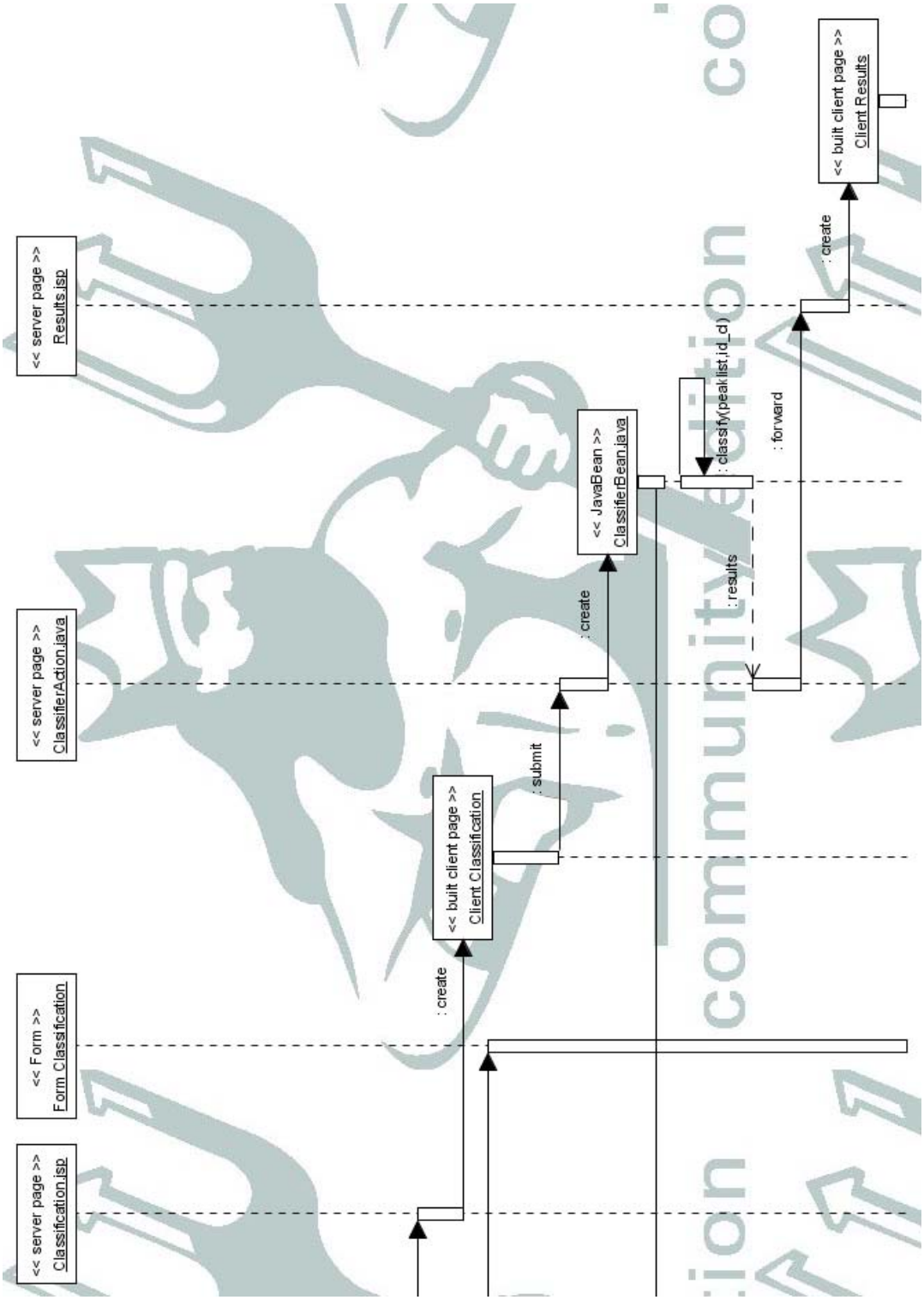


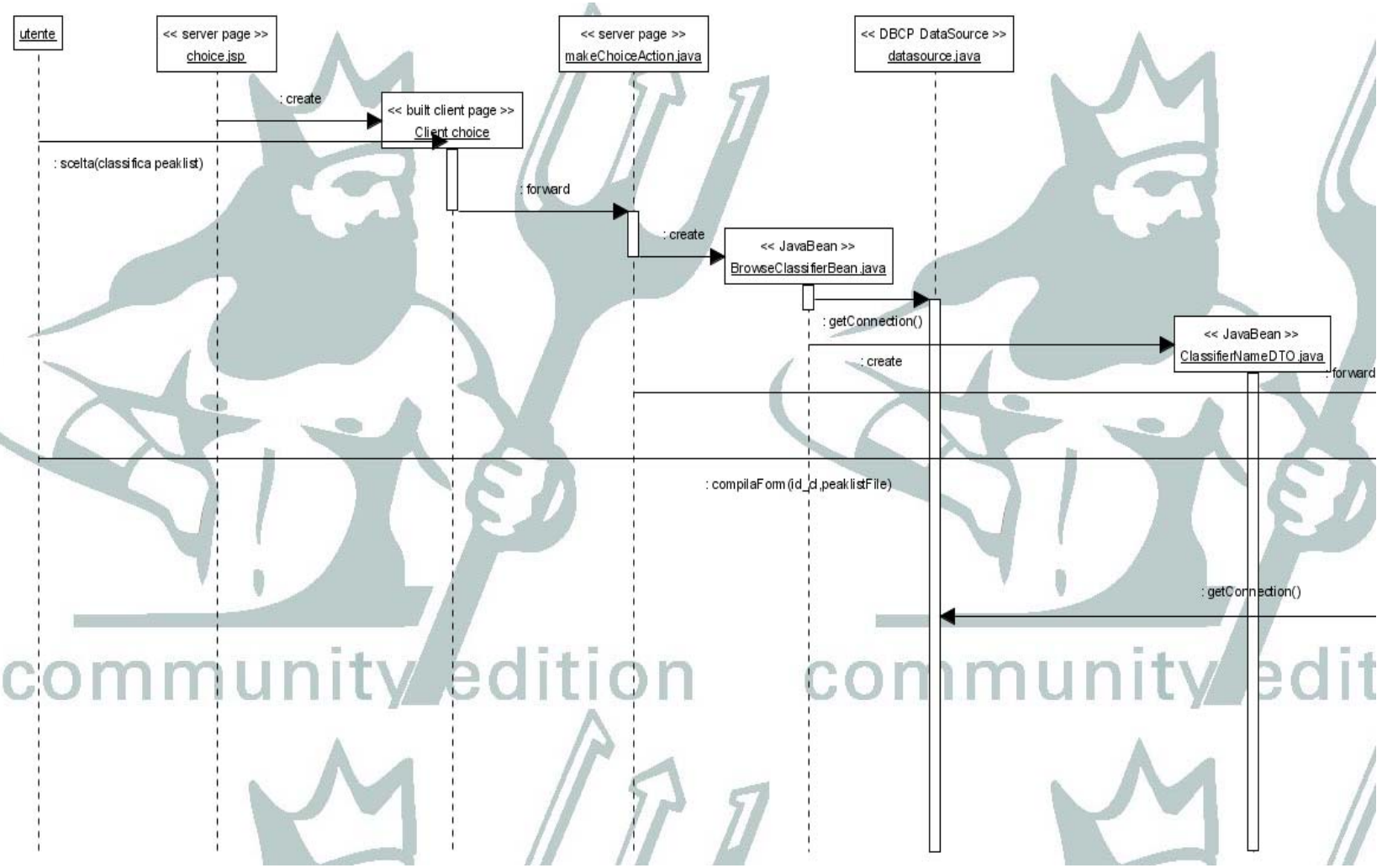


Sequence Diagram Design Level
Aggiungi Classificatore

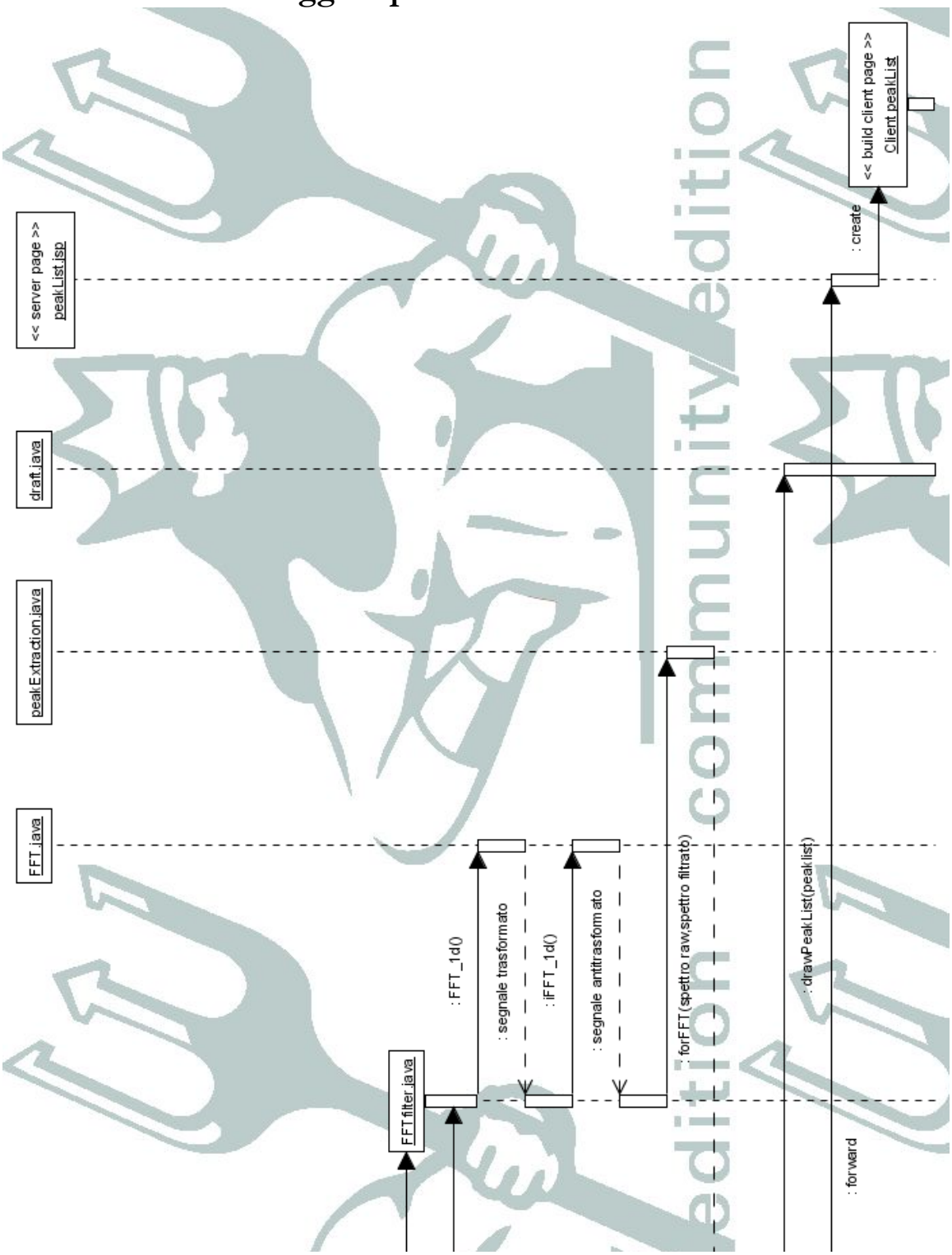


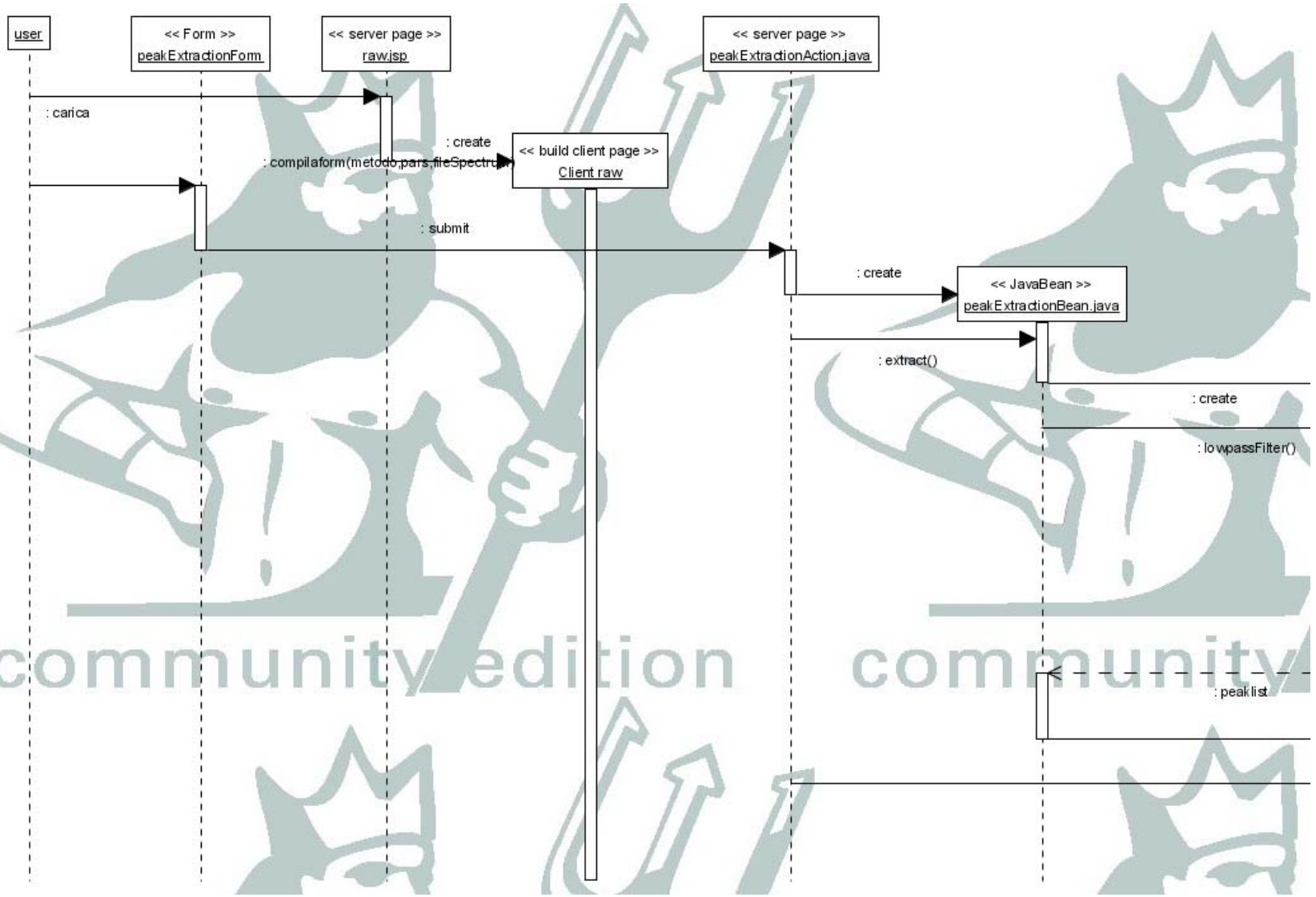
Sequence Diagram Design Level
Classifica peaklist



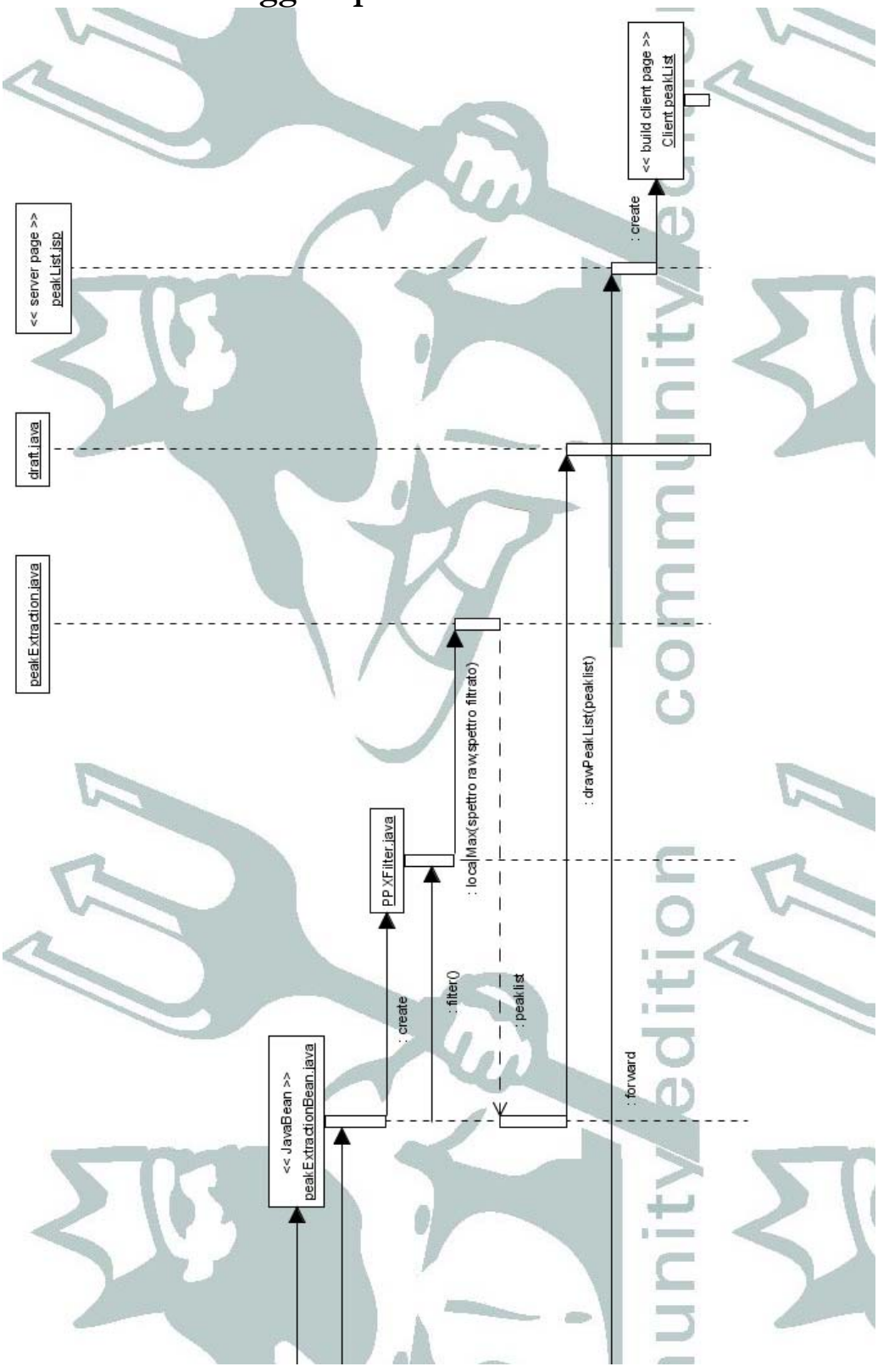


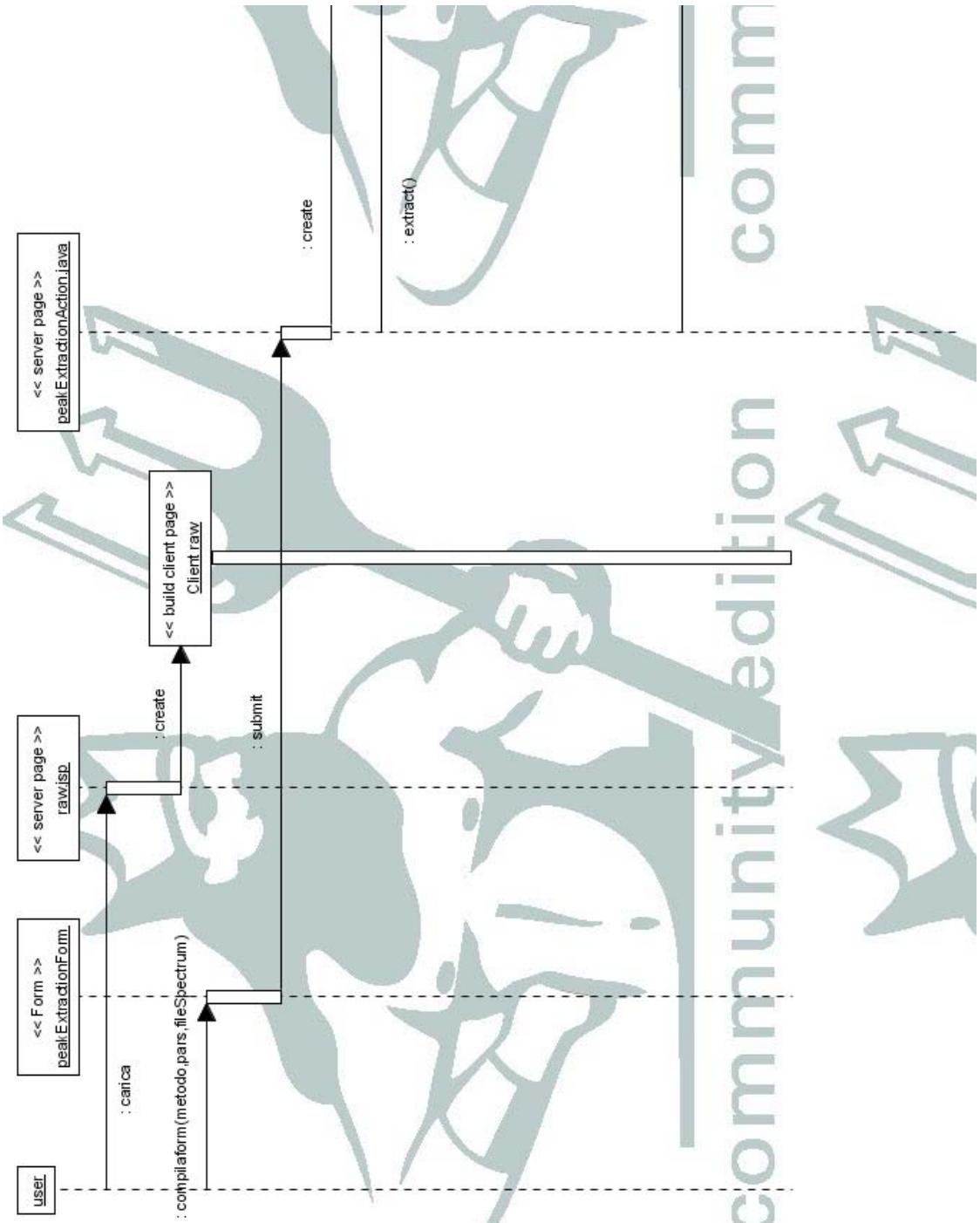
Sequence Diagram Design Level
Filtraggio Spettro Grezzo con FFT



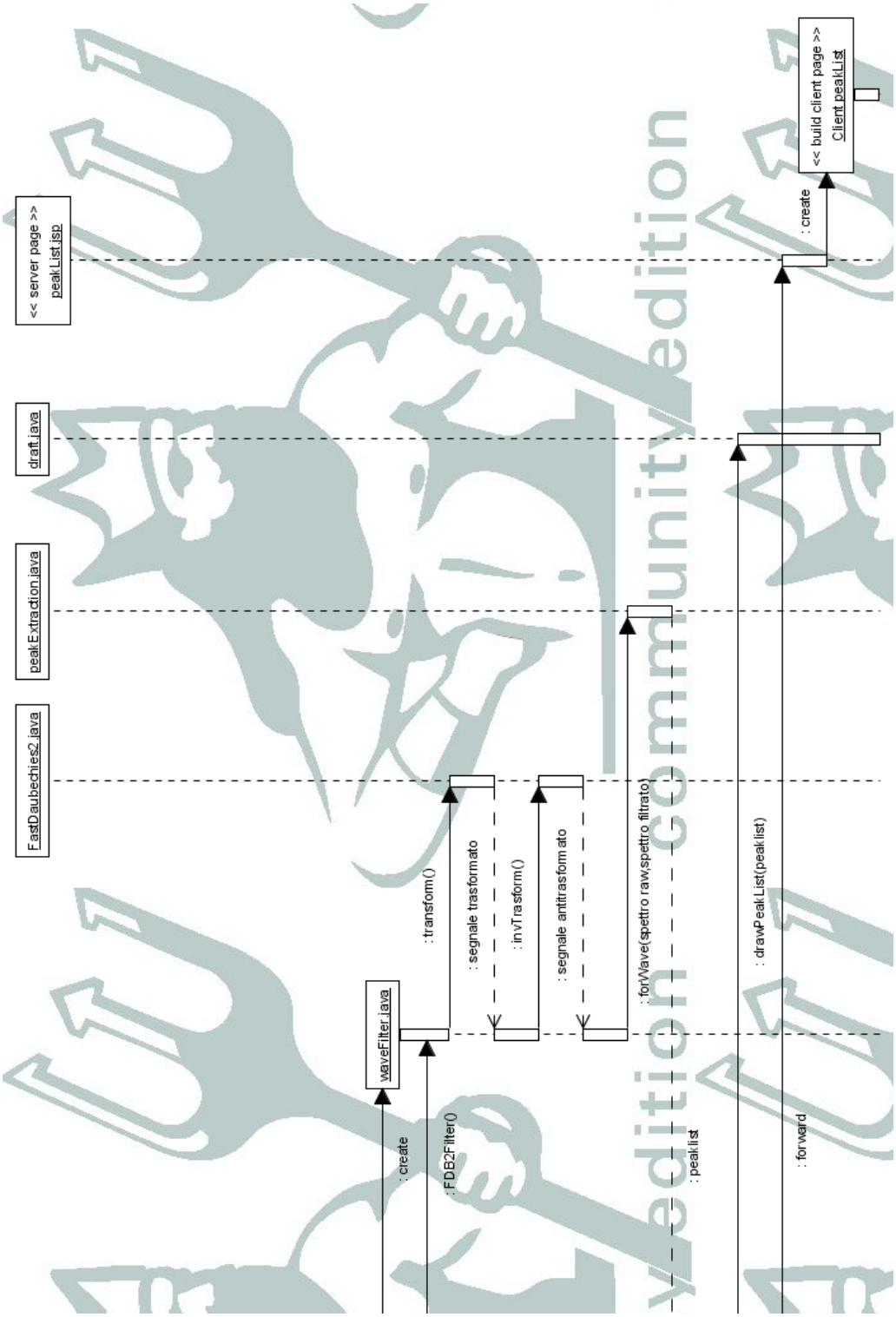


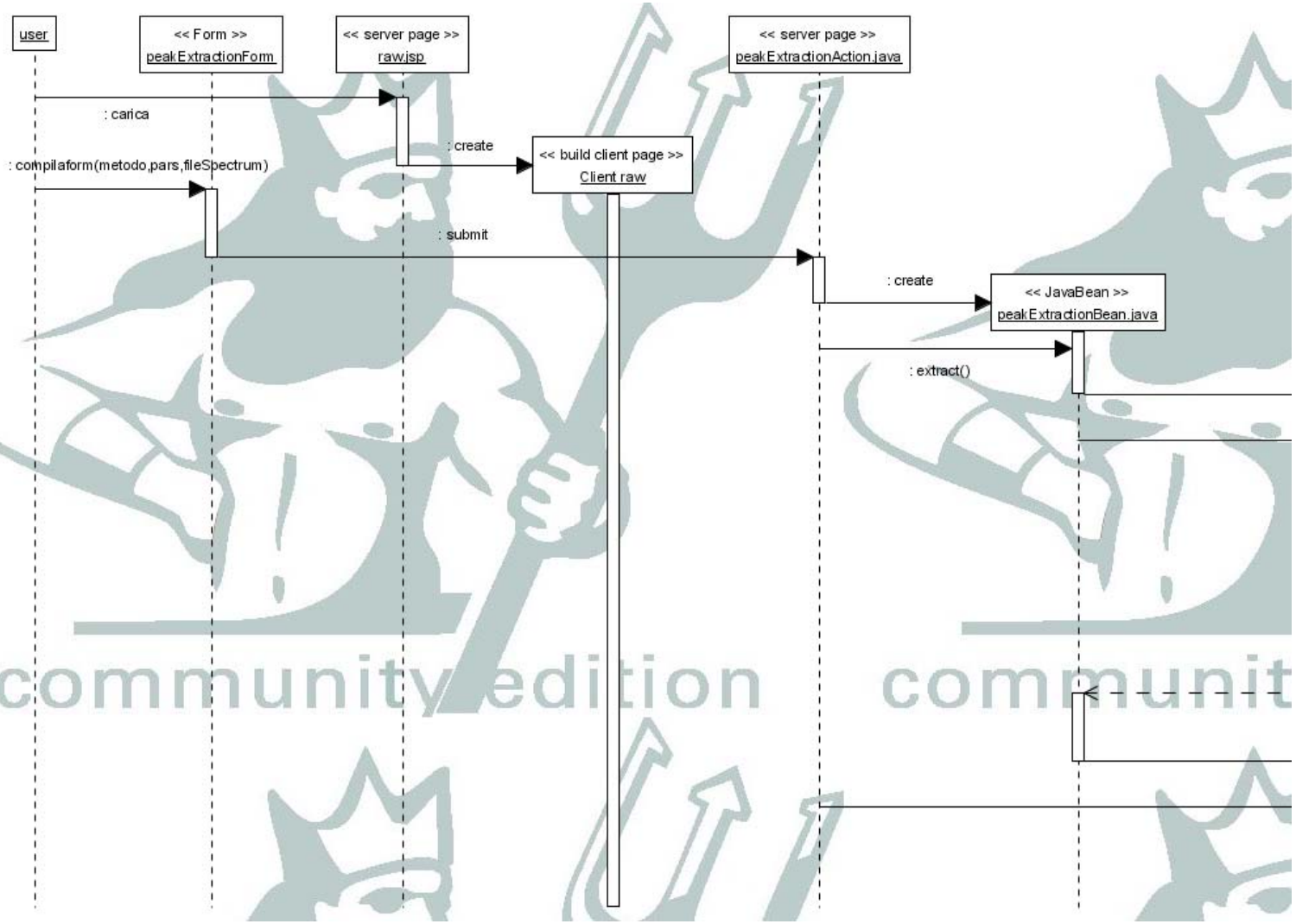
Sequence Diagram Design Level
Filtraggio Spettro Grezzo con PPX



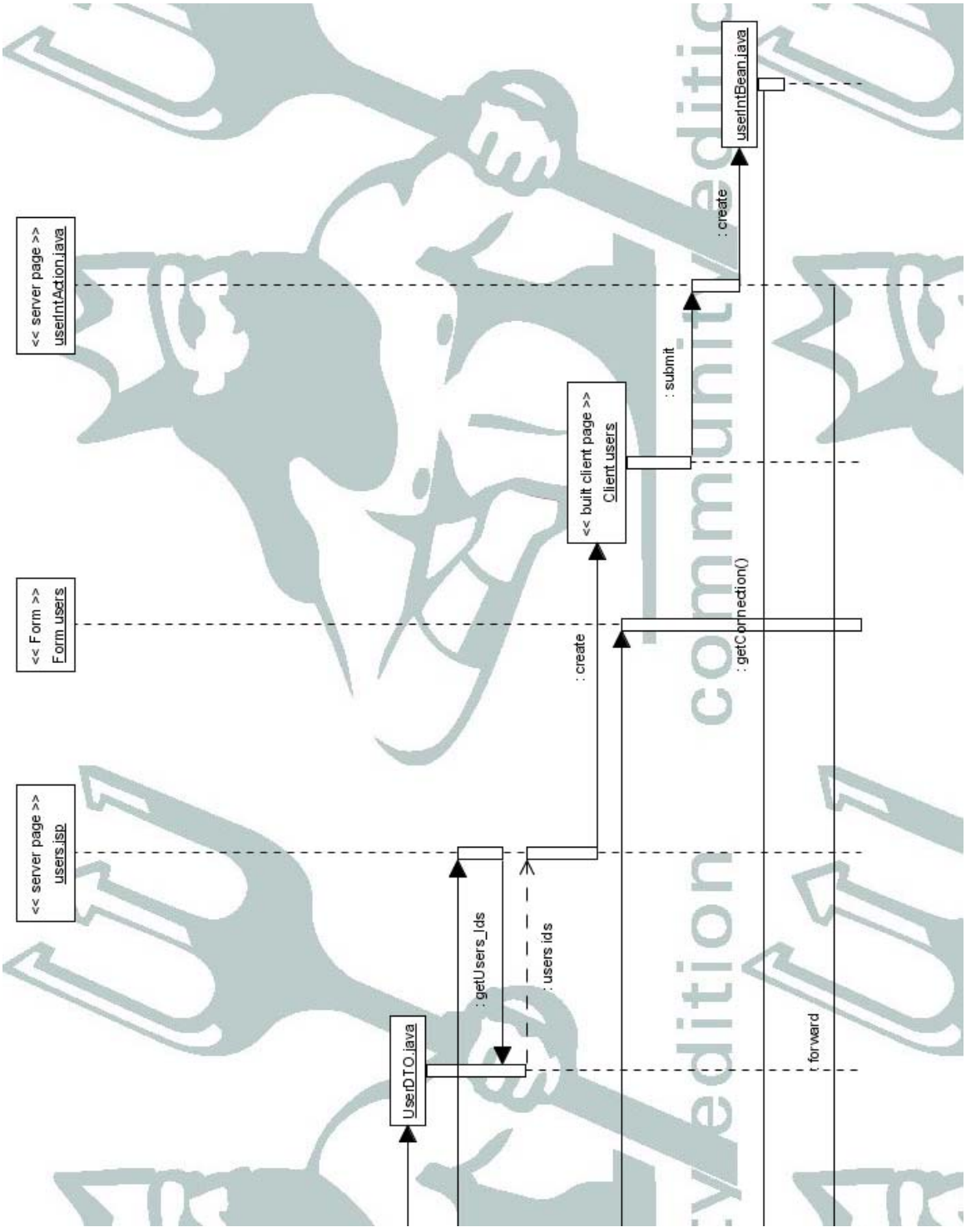

user
<< Form >>
peakExtractionForm
<< server page >>
raw.jsp
<< build client page >>
Client raw
<< server page >>
peakExtractionAction.java
: carica
: compilaform(metodo,pars,fileSpectrum)
: create
: submit
: create
: extract()

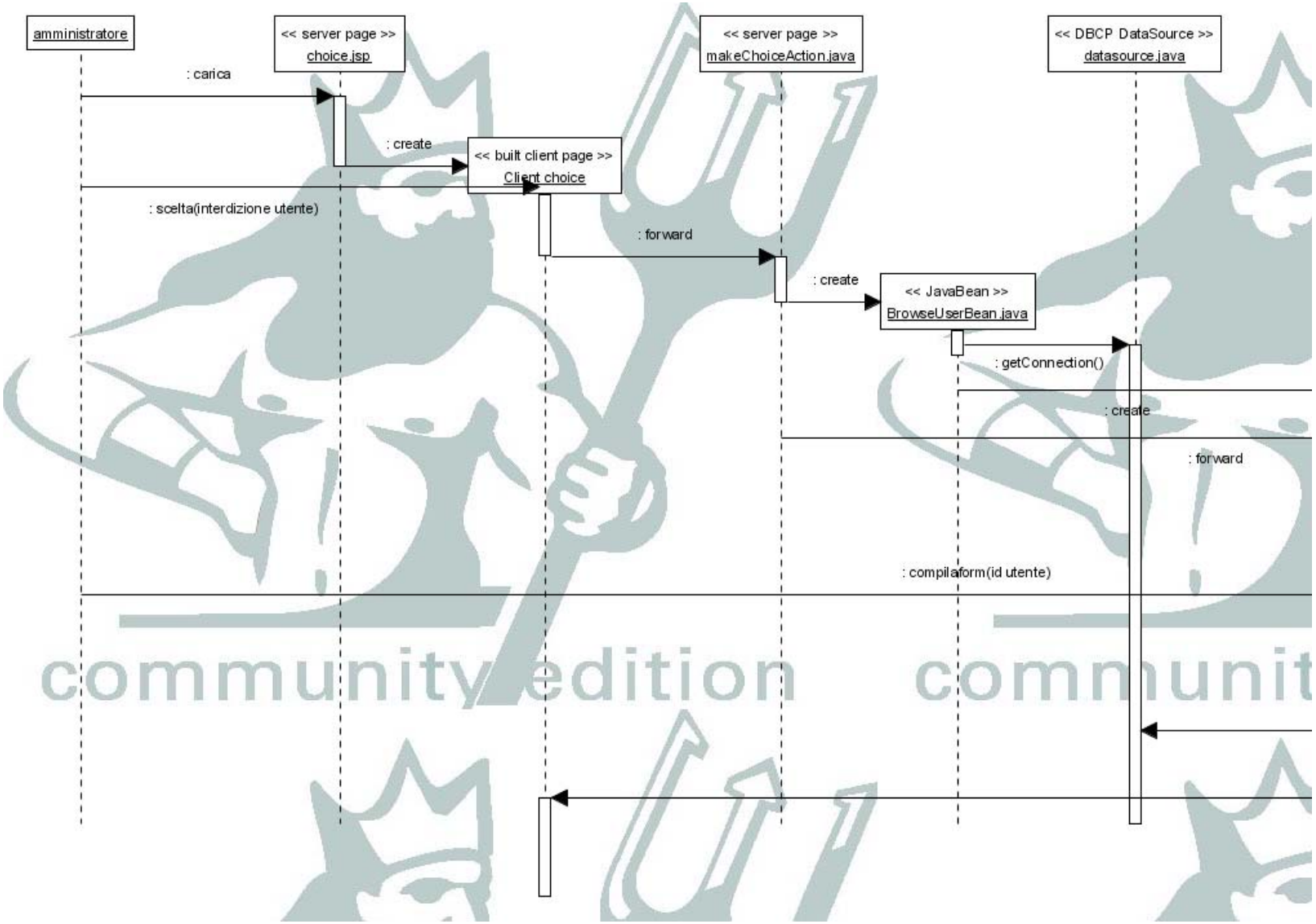
Sequence Diagram Design Level
Filtraggio Spettro Grezzo con Wavelet



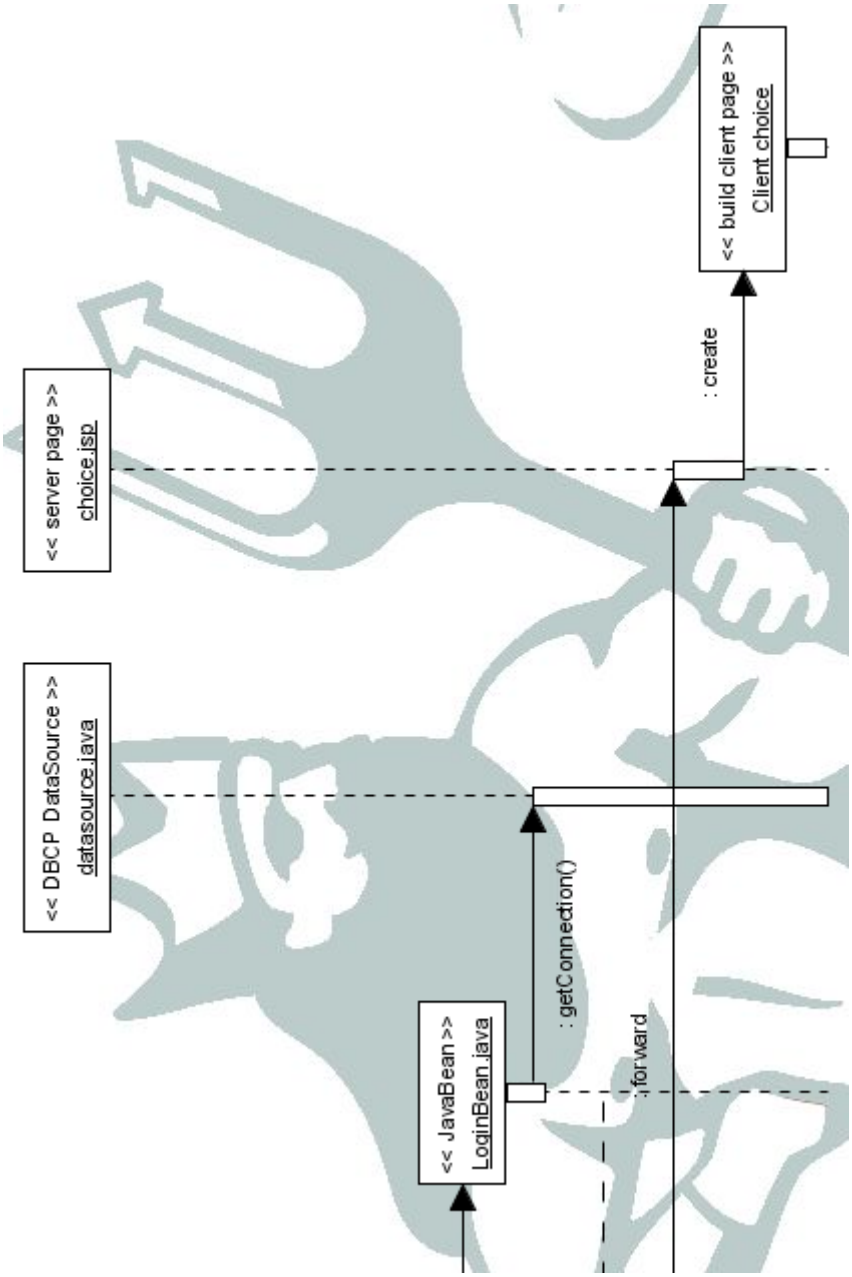


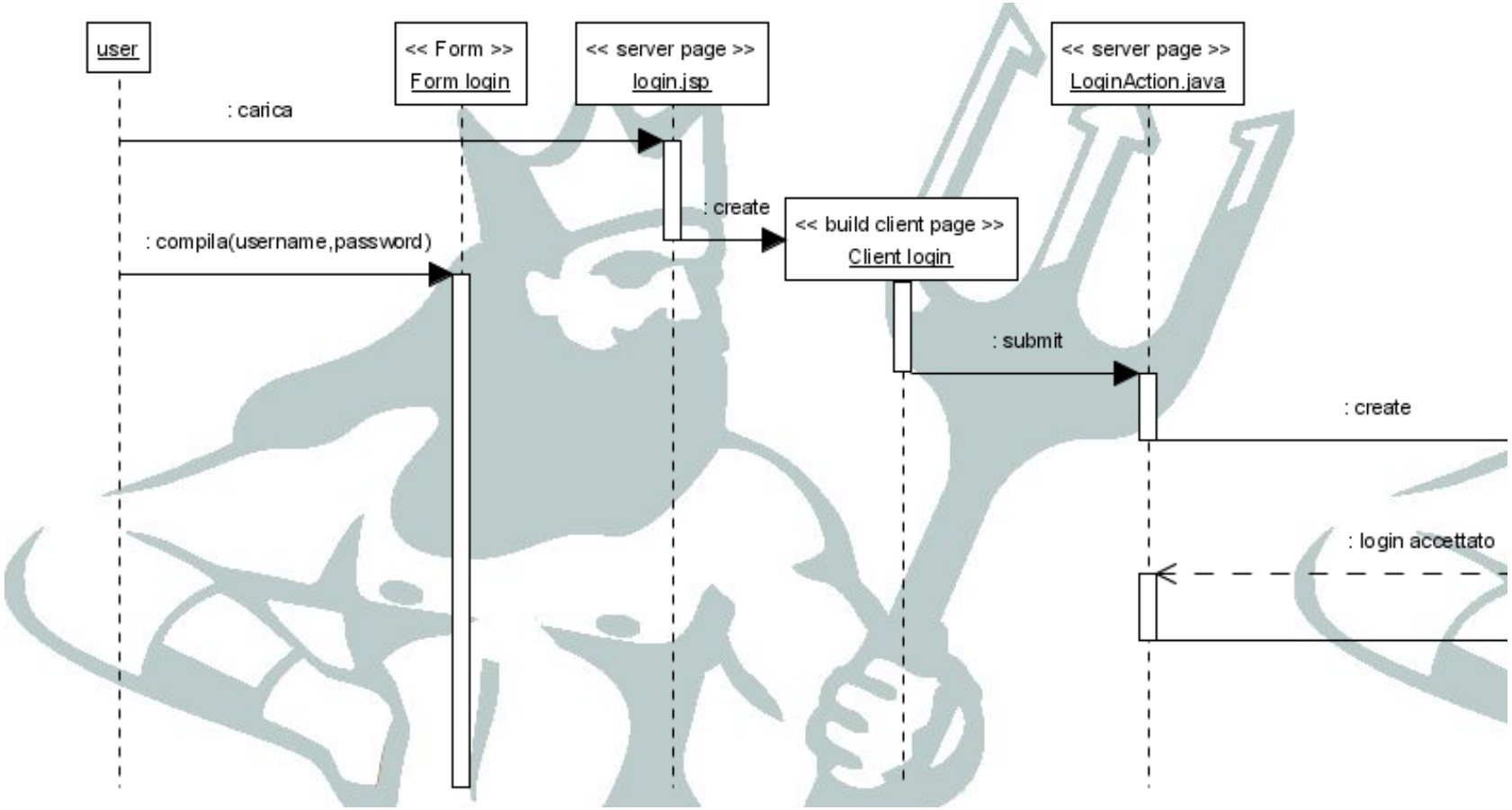
Sequence Diagram Design Level
Interdizione Utente



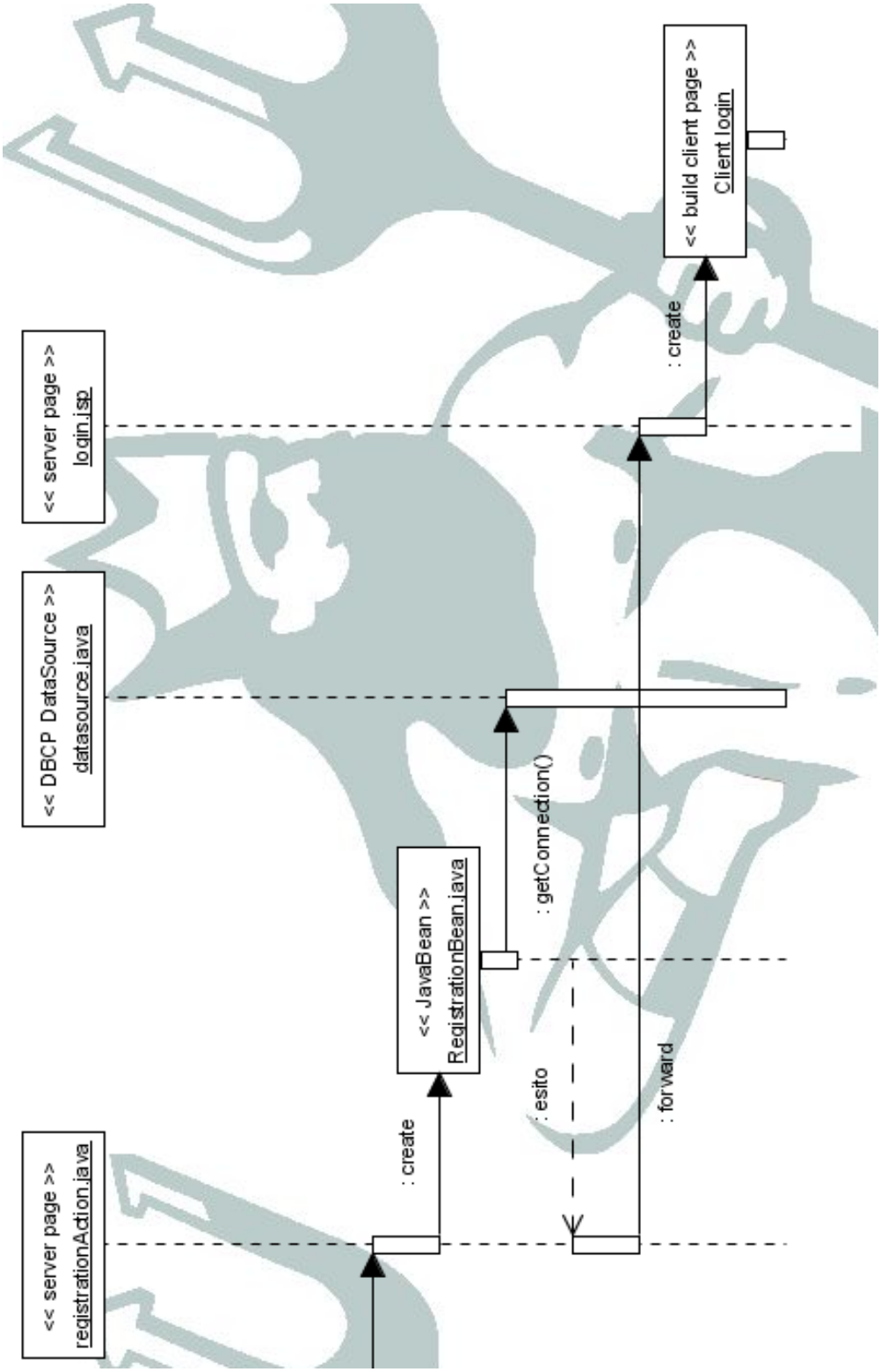


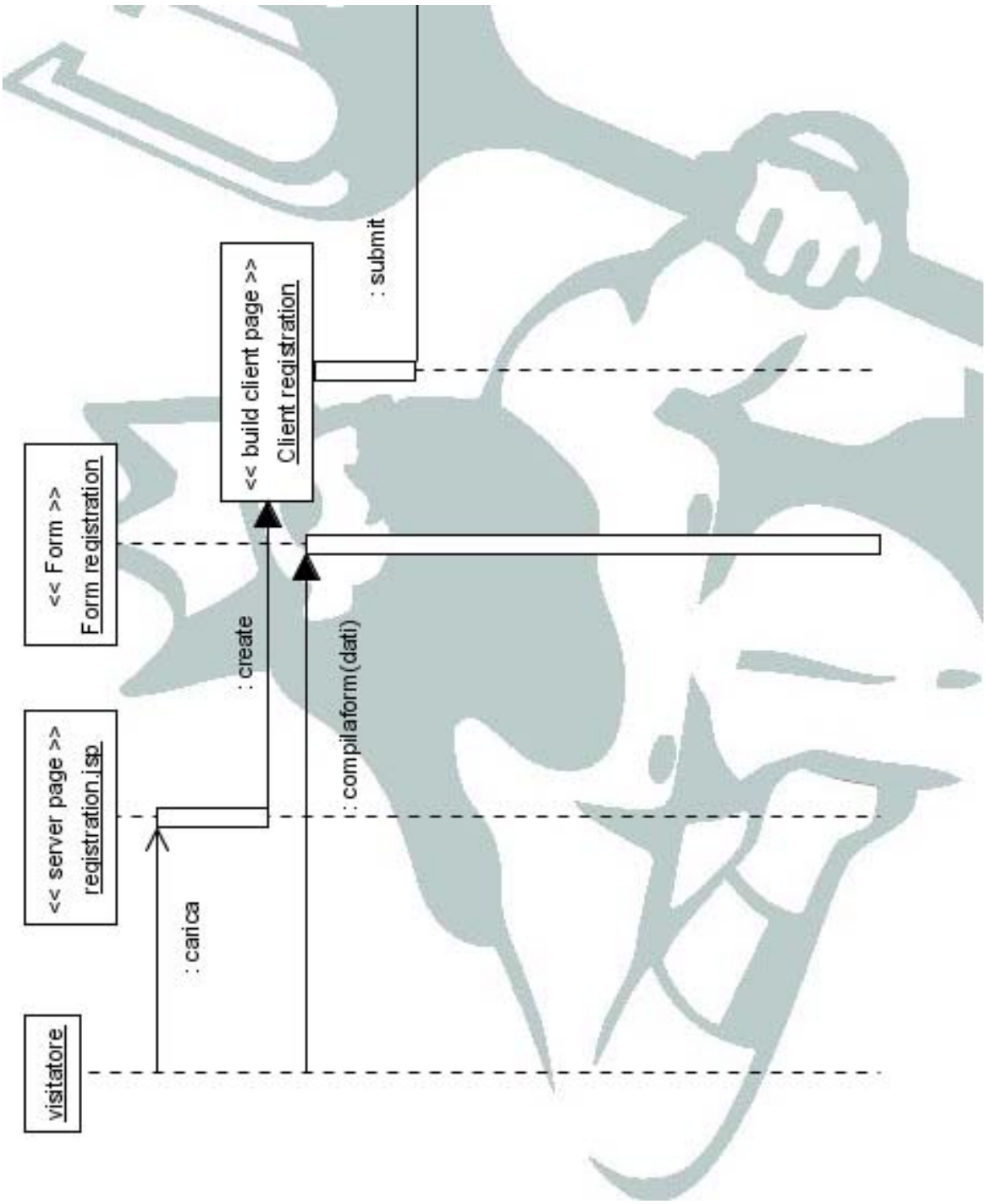
Sequence Diagram Design Level
Login



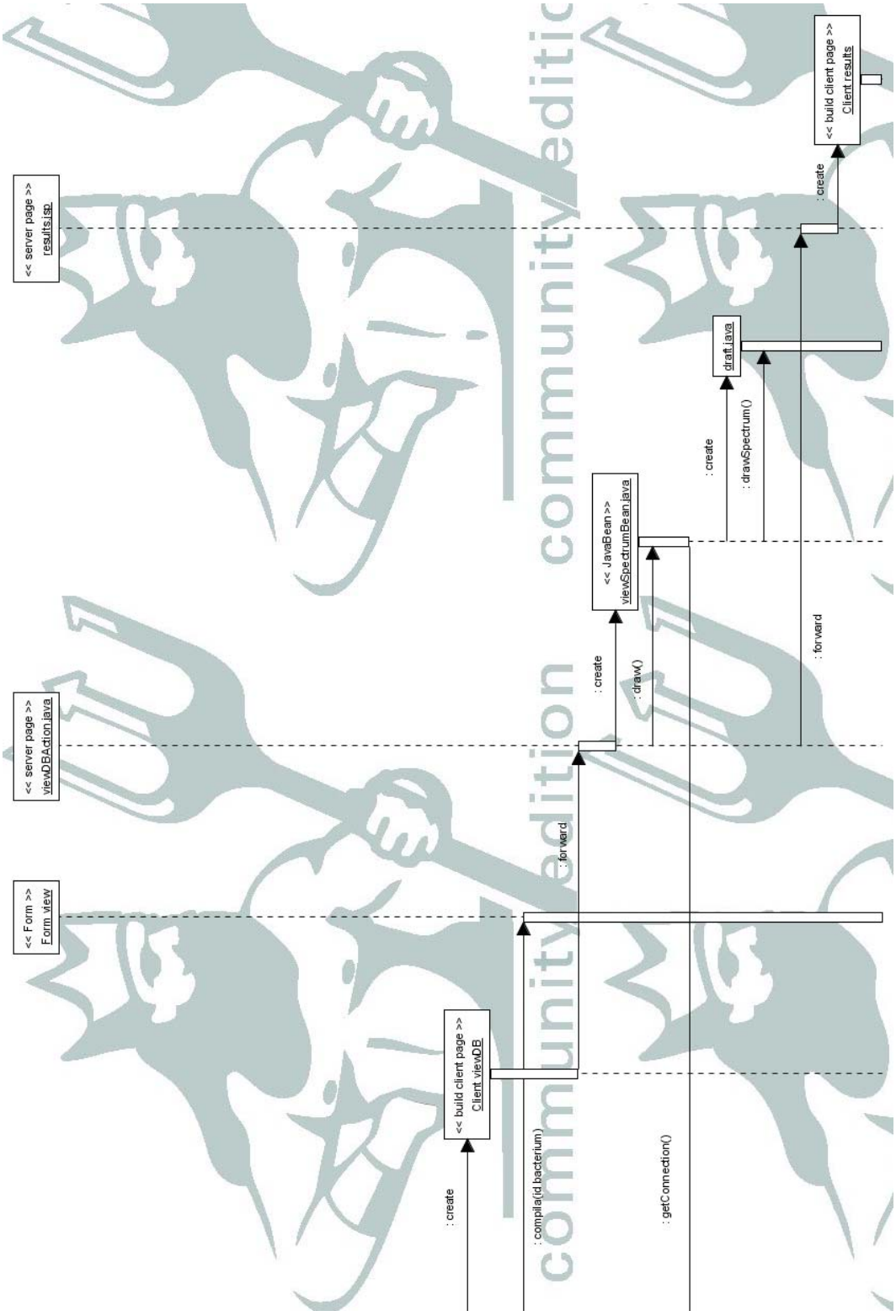

user
<< Form >>
Form login
<< server page >>
login.jsp
<< server page >>
LoginAction.java
: carica
: create
<< build client page >>
Client login
: compila(username,password)
: submit
: create
: login accettato

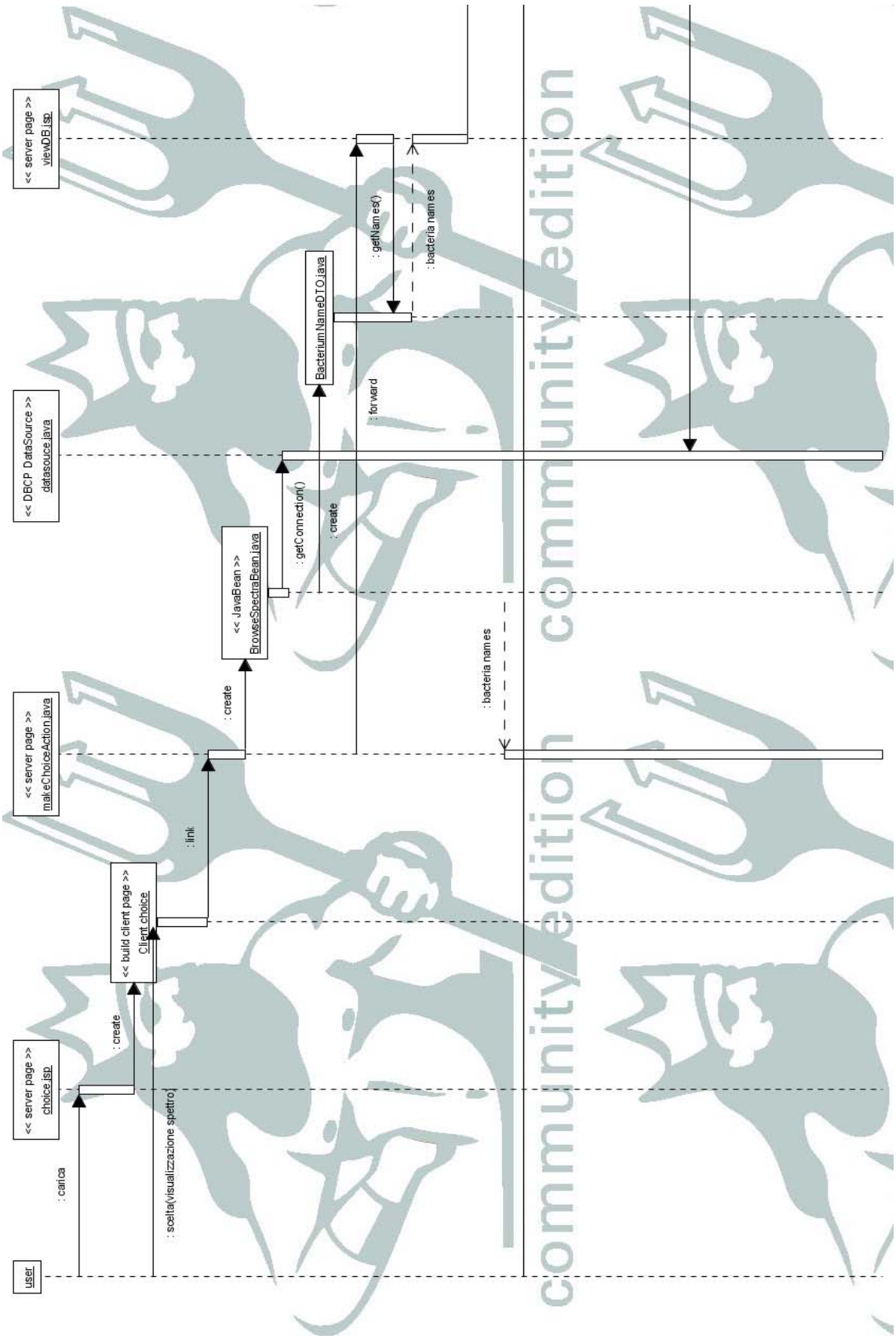
Sequence Diagram Design Level
Registrazione




visitatore
<< server page >>
registration.jsp
<< Form >>
Form registration
<< build client page >>
Client registration
: carica
: create
: compilaform(dati)
: submit

Sequence Diagram Design Level
Ricerca Spettro




user
<< server page >>
choice.jsp
<< build client page >>
Client choice
<< server page >>
makeChoiceAction.java
<< JavaBean >>
BrowseSpectraBean.java
<< DBCP DataSource >>
datasouce.java
BacteriumNameDTO.java
<< server page >>
viewDB.jsp
: carica
: create
: scelta(visualizzazione spettro)
: link
: create
: getConnection()
: create
: forward
: getNames()
: bacteria names
: bacteria names

Capitolo 6

CONCLUSIONI

E' stato fatto uno studio sul dominio applicativo, cercando di comprendere lo stato dell'arte sulla classificazione degli spettri di massa. Si è cercato di comprendere le caratteristiche degli spettri di massa in generale e, in particolare di quelle riguardanti i batteri. Si è proceduto quindi allo studio dei problemi del riconoscimento batterico e dei nuovi strumenti utilizzati in questo campo, come il Maldi-Tof. E' stata fatta anche una valutazione delle tecniche generali di classificazione, che ha comportato la ricerca di metodi per il trattamento degli spettri e per la loro caratterizzazione numerica in prospettiva di un'elaborazione algoritmica. Sono stati considerati i metodi di filtraggio dal rumore ed anche i metodi per la features extraction. Alla luce di tutte le considerazioni fatte si è cercato di realizzare un sistema di classificazione. Le prestazioni di questo sistema sono ancora insoddisfacenti, poiché gli manca un sistema di features selection, che permetta di evitare l'overfitting. Vi sono numerosi algoritmi di features selection, ed ognuno di essi va testato con il dataset e con tutti i possibili classificatori. Tale valutazione è onerosa, in termini di tempo, ma potrebbe migliorare decisamente le prestazioni dell'applicazione. Nel caso di prestazioni non ancora soddisfacenti, si può continuare a cercare migliori algoritmi di features extraction, oltre al metodo di calibrazione, e costruire architetture di classificazione più articolate, come i sistemi multiesperto o i classificatori gerarchici. Inoltre andrebbero provati degli algoritmi che calcolano una stima della correlazione tra gli spettri, che potrebbero rivelarsi utili per individuare somiglianze tra gli spettri batterici Per quanto riguarda la parte Web dell'applicazione, essa è ancora in fase di sviluppo e mancano ancora delle funzionalità come la parte amministrativa che è stata implementata a parte e andrebbe integrata. Inoltre tutto il sistema andrebbe provato utilizzando i metodi di testing standardizzati.

BIBLIOGRAFIA

[1]Visualization and analysis of molecular scanner peptide mass spectra
Markus Muller,Robin Gras
Swiss Institute of Bioinformatics, Geneva, Switzerland

[2]Feature selection and nearest centroid classification for protein mass spectrometry
Ilya Levner
Department of Computing Science, University of Alberta, Canada

[3]Class prediction and discovery using gene microarray and proteomics mass spectrometry data: curses,caveats,cautions
R.L. Somorjai
Institute for Biodiagnostics, National Reasearch Council Canada

[4]Fingerprint matching of E. coli Strains with Matrix-assisted Laser Desorption/Ionization Time-of-Flight Mass Spectrometry of whole cells using a modified correlation approach
Randy J. Arnold, James P. Reilly
Department of Chemistry, Indiana University, USA

[5]Improvement of peptides identification in proteomics with the use of new analytical and bioinformatic strategies
Tomas Baczek
Department of Biopharmaceutics and Pharmacodinamics, Medical University of Gdansk, Poland

[6]Compilation of a MALDI-ToF mass spectral database for the rapid screening and caracterization of bacteria implicated in human infectious diseases
Carrina J. Keys, Diane J. Dare et al.
Molecular Identification Services, National Collection of Type Cultures (NCTC), London, UK

[7]NIR and mass spectra classification: Bayesian methods for wavelet-based feature selection
Marina Vannucci, Naijun Sha and Philip J. Brown

[8]Rapid typing of bacteria using matrix-assisted laser desorption ionization time of flight mass spectrometry and pattern recognition software
John J. Bright, Martin A. Claydon, et al.
Bioanalytical Research Centre, Department of Biological Sciences, Manchester Metropolitan University, Manchester, UK

[9]Novelty Detection in mass spectral data using a support vector machine method
Christopher Tong and Vladimir Svetnik
Dept of Statistics, Purdue University, West Lafayette,IN

[10]Feature selection for high dimensional data: a Kolmogorov_Smirnov correlation based filter
Jacek Biesiada and Wlodzislaw Duch
Division of Computer Methods, Dept. of Electrotechnology, The Silesian University of Technology, Katowice, Poland

[11]Mining mass-spectra for diagnosis and biomarker discovery of cerebral accidents
Julien Prados, Alexandros Kalousis
Department of Computer Science, University of Geneva, Switzerland

[12]Feature extraction from mass spectra for classification
Alexandros Kalousis, Julien Prados, et al.
Department of Computer Science, University of Geneva, Switzerland

[13]Wavelet packet feature extraction method for effective classification of mass spectral data
Melissa McDermott and Frederick Warner

[14]Wavelet Tutorial
Robi Polikar

[15] Wavelet Methods for Time Series Analysis
Percival and Walden

[16]Bacterial Nomenclature Up-to-date
DSMZ-Deutsche Sammlung von Mikroorganismen und Zellkulturen GmbH, Braunschweig, Germany

[17]Estimating the generalization performance of a Support Vector Machine efficiently
Thorsten Joachims
University of Dortmund, Dortmund , Germany

[18]Machine learning, neural and statistics classification
D.Michie, D.J. Spiegelhalter

[19]Jakarta Struts for Dummies
Mike Robinson and Ellen Finkelstein
Wiley Publishing,Inc.

[20]Struts in Action
Ted Husted et al.
Manning

[21]Sviluppare applicazioni J2EE con Jakarta Struts
Alfredo Larotonda
http://www.mokabyte.it

[22]Improving Automatic Peptide Mass Fingerprint Protein Identification by Combining Many Peak Set
Thorsteinn Rognvaldsson et al.
School of Information Science, Halmstad University, Halmstad, Sweden

[23]Frequency analysis of Maldi-Tof spectra: noise filtering and signal detection
Thomas Kreitler et al.
Max Planck Institute for Molecular Genetics, Berlin