POLITECNICO DI MILANO

V Facoltà di Ingegneria

Laurea Specialistica in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

SERVIZI E APPLICAZIONI WEB DATA INTENSIVE BASATI SU GRANDI BASI DI CONOSCENZA IN EVOLUZIONE: IL CASO BIOINFORMATICO

Relatore: Prof. Marco Masseroli

Tesi di Laurea di:

Marco De Zorzi, matricola 734544

Anno Accademico 2009-2010

POLITECNICO DI MILANO

V Facoltà di Ingegneria

Laurea Specialistica in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

SERVIZI E APPLICAZIONI WEB DATA INTENSIVE BASATI SU GRANDI BASI DI CONOSCENZA IN EVOLUZIONE: IL CASO BIOINFORMATICO

Laureando: _____

(Marco De Zorzi)

Relatore: _____

(Prof. Marco Masseroli)

Anno Accademico 2009-2010

Ringraziamenti

Dedico questa Tesi alla mia famiglia: ringrazio mia madre, che mi ha sempre sostenuto e aiutato (oltre che sopportato) in tutto ciò che ho fatto, mia nonna, che mi è sempre stata vicina, e chi non c'è più, ma che fin quando è stato con me mi ha insegnato molto.

Ringrazio tutti i miei compagni in questa avventura al Politecnico, con cui ho condiviso divertimento e fatiche: grazie in particolare a Manuel, Massi, Yan, Davide, senza i quali questi anni non sarebbero stati così belli.

Un ringraziamento in generale va a tutti i miei amici, che mi hanno sopportato anche durante questo periodo di tesi in cui non sono stato molto presente.

Questi anni di studio sono stati un viaggio lungo e faticoso, ma mi hanno dato anche grandi soddisfazioni; ora che è giunto al termine posso dire che mi lascerà per sempre un bel ricordo: un grazie quindi anche a tutti i miei insegnanti, che mi hanno seguito e stimolato a proseguire.

Indice

1 Sommario **1**

2 Introduzione **5**

2.1 Ricerca genomica e proteomica 5

2.1.1 Tecnologie biomolecolari high-throughput 8

2.2 Terminologie e ontologie biomediche-molecolari 10

2.3 Banche dati genomiche e proteomiche 11

2.4 Analisi di arricchimento di annotazioni biomolecolari 12

2.4.1 Analisi statistica e correzione di test multipli 14

2.4.2 Stato dell'arte dei sistemi di analisi di arricchimento 16

2.5 Service Oriented Architecture (SOA) 17

2.5.1 SOAP 19

2.5.2 REST (Representational State Transfer) 20

3 Obiettivi della Tesi **21**

4 Definizione dei requisiti e scelte progettuali **25**

5 Materiali e metodi **31**

5.1 Service Oriented Architecture 31

5.1.1 RESTful Web Services 31

5.1.2 Apache CXF e JAX-RS 31

5.2 Web Server: Apache Tomcat 32

5.3 Strumenti per la gestione e l'elaborazione dei dati 33

5.3.1 PostgreSQL 33

5.3.2 pl/pgSQL 33

5.3.3 Java 34

5.3.4 JDBC 34

5.3.5 JDOM 34

5.4 Strumenti per la presentazione dei dati 34

5.4.1 Lato server: JSP e Servlet 35

5.4.2 Scripting dinamico: AJAX 35

5.4.3 Lato client: HTML e CSS 36
5.5 Annotazioni biomolecolari e strumenti per la loro analisi 36
5.5.1 Genomic and Proteomic Knowledge Base (GPKB) 36
5.5.2 Ontologizer 37
6 Risultati e discussione.....**41**
6.1 Architettura del sistema realizzato 41
6.2 Gestione ed elaborazione dei dati.....45
6.2.1 Uso dei dati del Genomic and Proteomic Data Warehouse (GPDW) 45
6.2.1.1 Utilizzo dei metadati.....45
6.2.1.2 Utilizzo dei numerosi dati biomolecolari.....46
6.2.1.3 Ottimizzazione delle query 49
6.2.2 Gestione dei dati dell'utente 50
6.2.2.1 Database del sistema sviluppato 50
6.2.2.2 Schema dei dati del sistema nel GPDW 52
6.2.3 Gestione di liste di ID di entità biomolecolari.....56
6.2.4 Gestione di annotazioni biomolecolari 57
6.2.5 Analisi statistiche: Ontologizer.....58
6.2.5.1 Estensione di Ontologizer.....58
6.2.5.2 Protocollo di aggiornamento di Ontologizer.....59
6.3 Servizi e applicazione web sviluppati 60
6.3.1 Servizi di look-up 60
6.3.2 Servizi per la gestione del database del sistema 62
6.3.3 Servizi di elaborazione dati 62
6.3.4 Chiamata ai servizi web.....63
6.3.5 Applicazione web 63
6.3.5.1 Client web 64
6.3.6 Configurazione del sistema.....64
6.3.7 Gestione delle sessioni.....66
6.3.7.1 Gestione delle sessioni in ambito REST.....67
6.4 Gestione dell'aggiornamento della base di conoscenza 68
6.4.1 Definizione del problema 68
6.4.2 Proposta per la risoluzione del problema.....69

6.4.3 Caso di utenti con dati indipendenti 72
6.4.3.1 Soluzione a requisiti minimi 74
6.4.3.2 Soluzione ottimizzata 75
6.4.3.3 Gestione degli errori 77
6.4.4 Caso di utenti con dati dipendenti 78
6.4.4.1 Gestione degli errori 79
6.4.5 Soluzione implementata 80
6.4.5.1 Algoritmo di ripristino dei dati memorizzati 82
6.5 Uso dei servizi e dell'applicazione web sviluppati 85
6.5.1 Servizi web 85
6.5.2 Applicazione web 86
6.5.2.1 Caricamento di liste di ID 86
6.5.2.2 Definizione dei parametri di analisi 88
6.5.2.3 Risultati dell'analisi 89
6.5.2.4 Gestione dei dati salvati 90
7 Valutazione dei risultati 93
7.1 Prestazioni del sistema realizzato 93
7.2 Usabilità del sistema 96
7.3 Risultati delle analisi di arricchimento implementate 96
8 Conclusioni 105
9 Sviluppi futuri 107
10 Bibliografia 109
11 Appendice 113

Capitolo 1

Sommario

La diffusione dei sistemi informativi web è in costante crescita, e si moltiplicano ogni giorno le applicazioni e i servizi che offrono ogni tipo di funzionalità agli utenti. In particolare l'architettura basata su servizi offre delle possibilità impensabili fino a pochi anni fa: integrazione automatica di singole funzionalità in processi complessi, interoperabilità fra sistemi eterogenei e grande scalabilità. I sistemi informativi web si trovano a gestire un numero sempre maggiore di dati: tali dati devono essere memorizzati e utilizzati in modo efficiente, per garantire dei tempi di risposta ridotti e minimizzare la quantità di spazio su disco occupata nei server. I dati necessari a grandi sistemi web sempre più frequentemente vengono integrati in basi di conoscenza che permettono una maggiore facilità di raccolta, gestione, analisi e distribuzione delle informazioni in essa contenute. Alcune di queste basi di conoscenza integrano dati omogenei, altre integrano dati provenienti da sorgenti diverse ed eterogenee: la gestione risulta in questo caso più complessa e richiede maggiore attenzione. Se le sorgenti dati forniscono degli aggiornamenti (più o meno frequenti), anche la base di conoscenza che le integra deve essere aggiornata conseguentemente: si parla in questi casi di basi di conoscenza in evoluzione. Un sistema informativo che si appoggi a una base di conoscenza di questo tipo deve essere in grado di gestire tale evoluzione. Inoltre, i sistemi informativi web che generano dati interagenti con la base di conoscenza devono integrare un meccanismo che permetta di mantenere la correlazione fra tali dati e la base di conoscenza durante la sua evoluzione. Se la mole di dati generata è elevata, come avviene in sistemi data intensive, il processo per garantire il mantenimento di tale correlazione tra due successive versioni della base di conoscenza può richiedere un tempo lungo, durante il quale è tuttavia necessario mantenere l'operatività del sistema web. In tali casi non sono pertanto adeguati i meccanismi normalmente utilizzati per sistemi basati su data warehouse, i quali prevedono l'aggiornamento off-line del data warehouse e il successivo veloce switch dalla versione precedente a quella aggiornata di tale data warehouse da parte dei sistemi che si appoggiano ad esso.

Scopo di questa Tesi è quindi la progettazione e realizzazione prototipale di servizi e applicazioni web di tipo data intensive, basati su basi di conoscenza che variano nel tempo, che siano efficienti in termini sia di spazio necessario per la memorizzazione dei dati sia di tempi di risposta agli utenti e che permettano di gestire l'evoluzione della base di conoscenza mantenendo la propria operatività.

Le problematiche sopra esposte vengono studiate nel caso bioinformatico, dove le architetture web descritte sono frequenti e rilevanti, e dove gli aspetti critici citati sono amplificati dalla presenza di dati molto numerosi che variano con una frequenza spesso molto elevata. In particolare viene progettato e realizzato un sistema di analisi di arricchimento di annotazioni biomolecolari basato su servizi web e sulla Genomic and Proteomic Knowledge Base, una grande base di conoscenza biomolecolare e biomedica che integra dati provenienti dalle maggiori banche dati mondiali del settore, realizzata nel Laboratorio di Basi di Dati del Dipartimento di Elettronica e Informazione del Politecnico di Milano.

I risultati raggiunti nel caso bioinformatico, che presenta criticità nei requisiti superiori rispetto a quanto avviene in altri domini, potranno essere direttamente applicati anche a sistemi informativi appartenenti ad altri ambiti.

Dopo il presente sommario, nel capitolo "*Introduzione*" viene introdotto l'ambito biomolecolare e bioinformatico considerato, gli scopi della ricerca genomica e proteomica, le tecnologie e le banche dati utilizzate; viene illustrato il procedimento utilizzato per analisi di arricchimento di annotazioni biomolecolari ed effettuata una rassegna dei principali strumenti software che offrono questa funzionalità; infine viene fatta una panoramica sulle architetture orientate ai servizi.

Gli obiettivi e le motivazioni che hanno portato allo sviluppo di questa Tesi sono illustrati nel capitolo "*Obiettivi della Tesi*".

Nel capitolo "*Definizione dei requisiti e scelte progettuali*" sono specificati i requisiti che in generale un sistema informativo web basato su una grande base di conoscenza in evoluzione deve soddisfare, e in particolare i requisiti del sistema sviluppato nello specifico caso bioinformatico di analisi di arricchimento di annotazioni biomolecolari. In tale capitolo sono anche illustrate le principali scelte effettuate nella progettazione di metodi per l'esecuzione di tali analisi.

Nel capitolo "*Materiali e metodi"* sono illustrati gli strumenti e le metodologie utilizzate per raggiungere gli scopi descritti e soddisfare i requisiti specificati.

Il capitolo "*Risultati e discussione*" illustra la progettazione e l'implementazione dei servizi e dell'applicazione web considerati, che includono un sistema per la gestione dell'evoluzione della base di conoscenza, descrive i risultati ottenuti e discute gli aspetti critici del lavoro svolto.

Nel capitolo "*Valutazione dei risultati*" sono valutate le prestazioni del sistema realizzato in relazione alle prestazioni di altri strumenti analoghi, ed è valutata l'usabilità complessiva del sistema da parte degli utenti; vengono inoltre confrontati i risultati biologici ottenuti nell'analisi di arricchimento implementata con quelli di altri strumenti che offrono la stessa funzionalità.

Nel capitolo "*Conclusioni*" vengono esposte le conclusioni raggiunte, che dimostrano la validità delle scelte progettuali fatte e del lavoro svolto per raggiungere gli obiettivi definiti.

Nel capitolo "*Sviluppi futuri*" vengono proposti eventuali miglioramenti o estensioni al sistema sviluppato realizzabili in futuro.

Il capitolo "*Bibliografia*" include i riferimenti a tutti i libri, articolo scientifici e siti web a cui ci si è riferiti nello svolgimento di questa Tesi.

In "*Appendice*" sono mostrati tutti i diagrammi dei database progettati non inclusi nei capitoli centrali della Tesi, oltre ai diagrammi di flusso e di sequenza delle operazioni principali del sistema sviluppato.

Capitolo 2

Introduzione

La bioinformatica è una disciplina scientifica dedicata alla risoluzione di problemi biologici a livello molecolare attraverso l'utilizzo di metodi informatici. Essa costituisce un tentativo di descrivere dal punto di vista numerico e statistico i fenomeni biologici fornendo ai risultati tipici della biochimica e della biologia molecolare un corredo di strumenti analitici e numerici. Vengono coinvolte, oltre all'informatica, la matematica applicata, la statistica, la chimica, la biochimica e nozioni di intelligenza artificiale.
Gli aspetti fondamentali di cui si occupa la bioinformatica sono principalmente:

- 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 al fine di creare un corpus di conoscenze relative alla frequenza di sequenze rilevanti, la loro evoluzione ed eventuale funzione
- organizzare le conoscenze acquisite a livello globale su genoma e proteoma in basi dati al fine di rendere tali dati accessibili a tutti, e ottimizzare gli algoritmi di ricerca dei dati stessi per migliorarne l'accessibilità.

L'obiettivo finale della bioinformatica è quello di scoprire la ricchezza di informazioni biologiche nascoste nella massa di dati a disposizione e di ottenere una visione più chiara sulla biologia degli organismi fondamentali.

2.1 Ricerca genomica e proteomica

La ricerca genomica si basa principalmente sullo studio delle analisi effettuate sul DNA (Acido Deossiribonucleico), che costituisce l'origine di tutte le informazioni necessarie per codificare le caratteristiche di un essere vivente e per gestirne lo sviluppo e il funzionamento dell'organismo.

Il DNA è la molecola (mostrata in figura 2.1), presente in tutte le cellule di un organismo, che si occupa di codificare la sintesi delle proteine all'interno di una cellula; le proteine sono gli elementi base di ogni struttura biologica, e ne definiscono le caratteristiche fondamentali. I geni sono una parte del DNA (circa l'1.5%) coinvolta nel processo di *trascrizione*, ovvero di trasmissione dell'informazione dal nucleo della cellula ai ribosomi, i quali sono organelli, situati nel citoplasma, che si occupano della sintesi delle proteine; il processo di trascrizione avviene tramite un altro acido nucleico, l'RNA (Acido Ribonucleico), che si occupa di trasportare le informazioni contenute nel DNA ai ribosomi. Il processo sopra descritto è soggetto inevitabilmente a variazioni dovute a problemi di natura esogena (quali temperature estreme, radiazioni, etc.) e di natura endogena (quali disturbi di trasmissione delle informazioni, mutazioni spontanee, etc.), e queste variazioni possono portare a patologie più o meno gravi.

La ricerca genomica è quindi nata per analizzare tali situazioni, studiare le malattie genetiche e considerare l'effetto delle terapie farmacologiche attuate/attuabili sulle cellule. Molte patologie sono infatti complesse, determinate da molti geni, come i tumori, e la conoscenza dell'intero genoma permette di identificare con più facilità i geni coinvolti e osservare come questi interagiscono nei vari processi biologici.

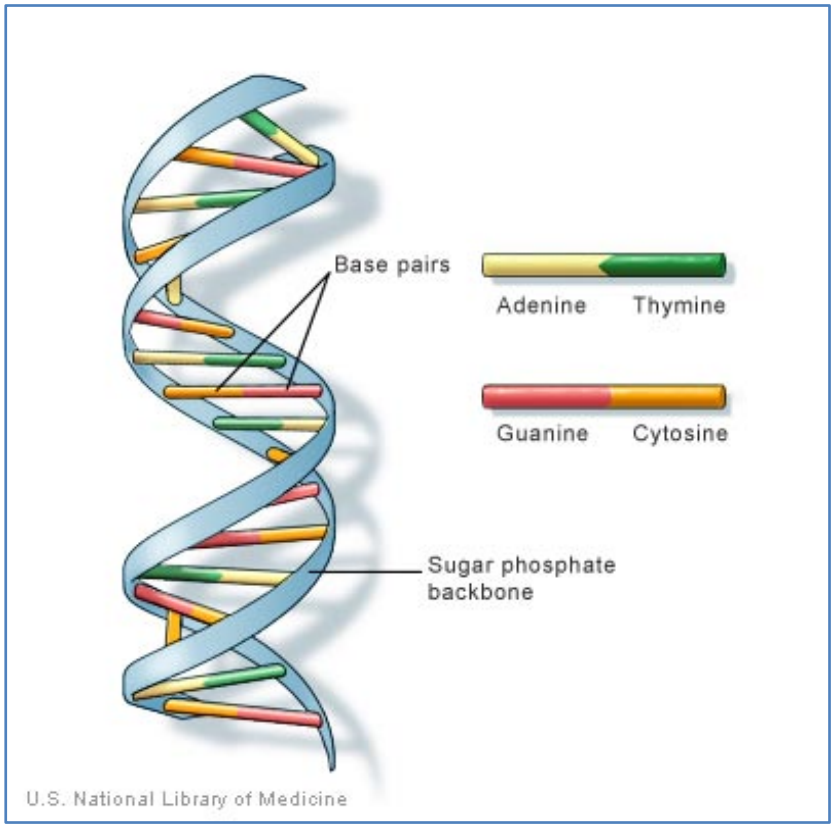


Figura 2.1. Struttura del DNA

La genomica nacque negli anni ‘80, quando fu sequenziato il primo genoma di un virus, il fago Φ-X174. Il primo sequenziamento del genoma di un organismo vero e proprio fu completato nel 1995 e si trattava di un batterio, Hemophilus influenzae (1,8 milioni di paia di basi). Il primo sequenziamento del genoma umano (circa 3 miliardi di basi, 30.000 geni) fu pubblicato nel 2001, pari al 90% della sequenza e ancora con notevoli probabilità di errori. Una sequenza accurata al 99,99% e pari al 99% del genoma umano è stata pubblicata nel 2003. Nel genoma è racchiusa tutta l'informazione genetica necessaria alla vita della cellula; esso è uguale in tutte le cellule di un organismo ed è uguale per il 99% degli individui di una specie.

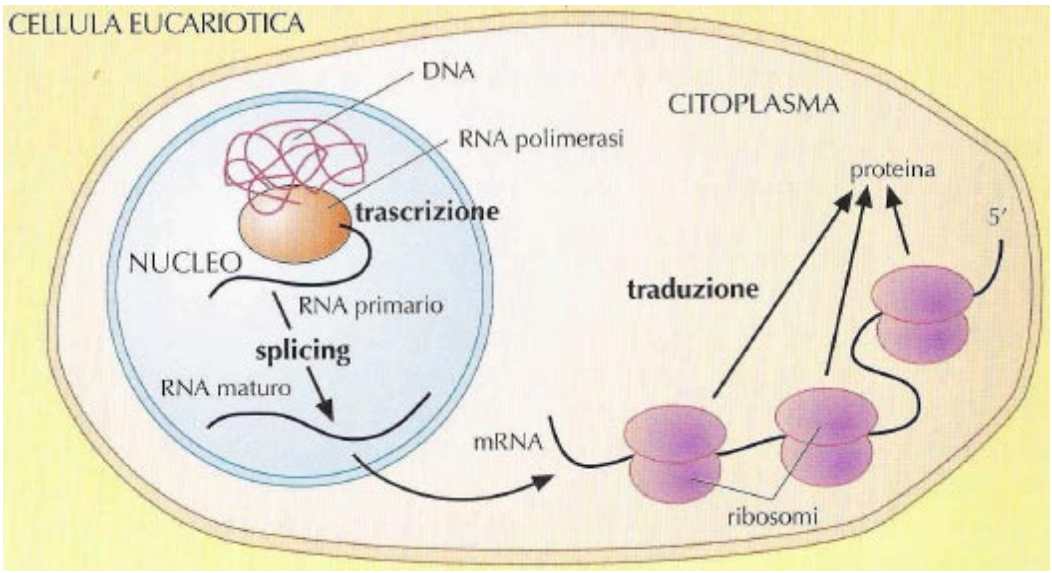


Figura 2.2. Trascrizione e traduzione in cellule eucariote

La regione codificante (cioè che produce proteine) rappresenta solo il 3% del genoma, mentre più del 50% sono sequenze ripetute. Il processo di sintesi proteica si divide in due fasi (mostrate in figura 2.2): la fase di trascrizione, in cui le informazioni contenute nel DNA vengono trascritte in una molecola di RNA e la fase di traduzione, che consente la sintesi di proteine a partire dall'RNA. Il processo di traduzione avviene all'interno dei ribosomi, ovvero le strutture specifiche della cellula adibite alla sintesi proteica. Il processo di biosintesi proteica è illustrato schematicamente in figura 2.3.

Strutturalmente una proteina è una sequenza di numerosi aminoacidi. Le proteine svolgono diversi compiti, essenziali, per la sopravvivenza della cellula stessa e dell'organismo. Il proteoma è l'insieme di tutti i possibili prodotti proteici (sequenze amminoacidiche) espressi in una cellula di un organismo e derivanti da diversi trascritti.

La proteomica consiste quindi nell'identificazione sistematica di proteine e nella loro caratterizzazione rispetto a struttura, funzione, attività, quantità e interazioni molecolari. Tale termine è stato coniato in analogia al termine genomica, disciplina rispetto alla quale la proteomica rappresenta il passo successivo, essendo molto più complessa. Infatti, mentre il genoma è un'entità pressoché costante, il proteoma differisce da cellula a cellula ed è in continua evoluzione nelle sue continue interazioni con il genoma e l'ambiente. Un organismo ha espressioni proteiche radicalmente diverse a seconda delle varie parti del suo corpo, nelle varie fasi del suo ciclo di vita e nelle varie condizioni ambientali.

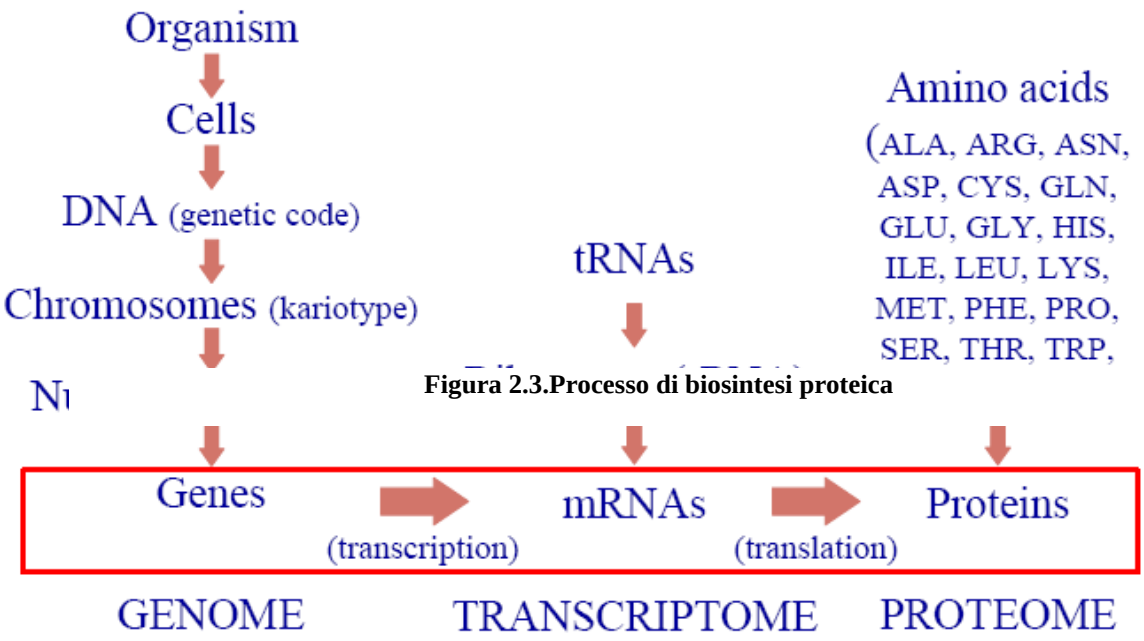


Figura 2.3.Processo di biosintesi proteica

Figura 2.3. Schema dei processi di trascrizione e traduzione

2.1.1 Tecnologie biomolecolari high-throughput

Dopo che nel 2001 è stato completato il sequenziamento del genoma umano, la ricerca si è spostata sull'analizzare singoli geni per capirne il funzionamento. Data la complessità dei processi biologici, però, solitamente più geni sono coinvolti e interagiscono nella regolazione di un processo. Da qui nasce la necessità dei ricercatori di analizzare più geni contemporaneamente.

Grazie allo sviluppo delle tecnologie biomolecolari, attualmente esistono strumenti che consentono di effettuare esperimenti su un numero elevatissimo di geni contemporaneamente, definite tecnologie high-throughput.

Una delle tecniche più utilizzate è quella dei microarray, che permette di analizzare decine di migliaia di geni in parallelo e di generare conseguentemente una grande quantità di dati risultanti dall'esperimento. Esistono due tipi di microarray: quelli costituiti da speciali vetrini da laboratorio, sui quali vengono posizionate numerose sequenze di geni, realizzabili in qualunque laboratorio di biologia molecolare; gli oligochip commerciali di Affymetrix [1], veri e propri chip di silicio costruiti con tecniche litografiche, sui quali vengono depositati oligonucleotidi, cioè brevi sottosequenze di geni. Solitamente gli esperimenti sono basati sul confronto tra geni in una condizione di test (ad esempio prelevati da una cellula trattata con farmaci) e una condizione di controllo (ad esempio prelevati da una cellula non trattata), ovvero lo studio dei cambiamenti dell'espressione genica. Se ad esempio confrontiamo cellulare sane e patologiche, una maggior presenza di un gene nelle condizioni patologiche farà ipotizzare una correlazione tra quel gene e la patologia considerata. Lo stesso tipo di analisi può essere fatta sulle proteine codificate da un determinato gene, ovvero analizzare se la proteina codificata è in maggiore quantità in condizione di test rispetto a quella di controllo.

L'output di un esperimento con microarray è quindi una lista di identificativi di geni risultati sovra-espressi o sotto-espressi rispetto alla condizione di test.

In figura 2.4 è mostrato un esempio di lista di identificativi con a fianco il livello di sovra-espressione (1) o sotto-espressione (-1) rispetto a una condizione di test.

AI805289	-1
AI907485	-1
AI983494	-1
AJ223321	-1
AK000256	1
AK000684	-1
AK001309	1
AK001971	1
AK002049	1
AK021514	-1
AK021863	1
AK021947	1
AK022078	-1

Figura 2.4. Esempio di lista di identificativi biomolecolari

Questa lista è poco utile se non si è in grado di associare a ogni identificativo informazioni sui processi in cui è coinvolto, sulle patologie associate e sulle sue

funzioni. È quindi necessario ricercare annotazioni (associazioni fra gli identificativi di entità biomolecolari e identificativi di termini biomedici presenti nelle banche dati) ed effettuare un'analisi statistica, in modo da fornire un'interpretazione dei risultati significativa.

2.2 Terminologie e ontologie biomediche-biomolecolari

In ambiti molto estesi e complessi come quello medico-biologico, in cui è necessario poter identificare con precisione concetti diversi, per condividere e utilizzare conoscenza si ricorre ai vocabolari controllati (o terminologie). Si tratta di particolari raccolte di termini, precisi e universalmente comprensibili, che definiscono e identificano dei concetti in modo univoco e inequivocabile. Questi vocabolari si dicono controllati perché vengono scritti e mantenuti aggiornati da un gruppo di persone preposte, i curatori. Nella biologia molecolare si sta adottando sempre più spesso l'uso di tali vocabolari: mentre in passato ogni entità biomolecolare veniva annotata con attributi descrittivi propri del singolo ricercatore, che potevano dare origine a sinonimi, omonimi e inconsistenze, quando la conoscenza da essi rappresentata veniva condivisa con la comunità scientifica, oggi si tende sempre più a usare termini standard, universalmente comprensibili. Tali termini permettono di contare le occorrenze delle entità biomolecolari in un certo insieme associate a ogni specifico termine e quindi la realizzazione di procedure computazionali automatiche.

Un esempio di vocabolario controllato è OMIM, che descrive le patologie ereditarie, ma esistono anche altri vocabolari che descrivono le caratteristiche funzionali delle proteine, dei corrispondenti geni e la loro localizzazione subcellulare e tessutale.

Tutte le annotazioni dei vocabolari controllati sono disponibili pubblicamente nelle rispettive banche dati (PFAM, KEGG, OMIM, etc.). L'utilizzo di questi vocabolari consente ai ricercatori e laboratori di analisi, durante gli esperimenti, di integrare le informazioni presenti sulle diverse banche dati.

Le ontologie sono composte da vocabolari controllati, che descrivono i concetti di un particolare dominio, e da una rete semantica, che descrive le relazioni esistenti tra i vari concetti. La struttura di un'ontologia è un grafo diretto aciclico (DAG), in cui i nodi rappresentano i concetti di un vocabolario controllato, mentre gli archi rappresentano

delle relazioni fra essi. Ogni termine del vocabolario è collegato con altri termini attraverso relazioni che specificano la conoscenza del particolare dominio in cui sono coinvolti tali termini. Un concetto generale può contenere molti termini che possono essere sinonimi o che possono riferirsi a differenti domini in cui è presente tale concetto. Per questo motivo, le ontologie tendono spesso ad avere una struttura gerarchica con termini generici ai livelli alti della gerarchia e termini specifici ai livelli bassi, collegati fra loro da diversi tipi di relazioni. Un esempio di ontologia in ambito genomico-molecolare è la Gene Ontology (GO) che consente la descrizione delle funzioni di geni e proteine di tutti gli esseri viventi e comprende tre vocabolari controllati relativi a funzioni molecolari, processi biologici e componenti cellulari.

2.3 Banche dati genomiche e proteomiche

Ad oggi diversi gruppi di ricerca pubblici e privati lavorano al sequenziamento e all'analisi del genoma, e con lo sviluppo di nuove tecniche di sequenziamento automatico (come i microarray) un vasto ammontare di dati viene prodotto e necessita di tecniche di analisi che consentano di estrapolarne conoscenza.

Questi dati sono quindi raccolti nelle banche dati biomolecolari, che riuniscono le informazioni su determinati settori di ricerca e offrono un pubblico accesso a tali informazioni sul web.

Esistono diverse banche dati contenenti:

- sequenze nucleotidiche
- dati di mappatura del genoma
- profili di espressione (2S-SDS PAGE, DNA chips)
- sequenze di proteine
- strutture 3D di acidi nucleici e proteine
- dati genotipici e trascritti
- dati metabolici
- annotazioni fenotipiche e funzionali
- informazioni bibliografiche

Ad oggi si contano 1.230 banche dati [2] (la crescita nel tempo è visibile in fig. 2.5) che in base alla tipologia di dati che contengono possono essere suddivise in:

- primarie: DNA, proteine, strutture 3D

- derivative (o specializzate): RNA, pathway, dati di microarray, …

Per primarie si intendono banche dati di sequenze biomolecolari che contengono le informazioni per identificare una sequenza dal punto di vista della specie e della funzione. Le banche dati specializzate, invece, arricchiscono le precedenti informazioni con dati tassonomici, biologici, fisiologici, funzionali, etc.

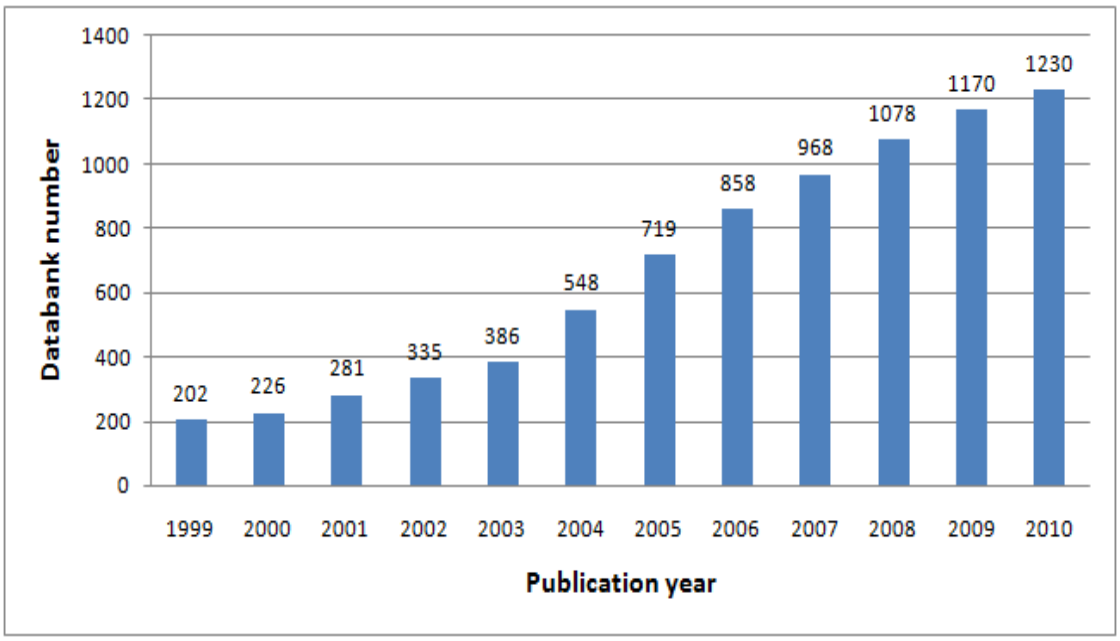


Figura 2.5. Crescita del numero di banche dati biomolecolari e biomediche nel tempo

Altro aspetto fondamentale è la frequenza di aggiornamento, poiché alcune banche dati vengono aggiornate giornalmente (KEGG [3]) mentre in altri casi sono aggiornate trimestralmente (InterPro [4]). L'aggiornamento può interessare non solo i dati contenuti al suo interno ma anche il formato dei dati o la tipologia di dati forniti.

2.4 Analisi di arricchimento di annotazioni biomolecolari

Per dare significato alle analisi sperimentali è necessario quindi recuperare le annotazioni relative agli identificativi delle entità biomolecolari coinvolte ed effettuare un'analisi di arricchimento, cioè un'analisi statistica di annotazioni di entità biomolecolari differenzialmente espresse in un esperimento [5]. L'analisi si svolge secondo questo procedimento: si parte da due insiemi di identificativi di entità biomolecolari derivanti da esperimenti svolti in laboratorio (spesso utilizzando

tecnologie a microarray), l'insieme detto *master* (o *population set*), che contiene tutte le entità biomolecolari che sono presenti sul microarray, e l'insieme detto *target* (o *study set*), che comprende solo geni o proteine differenzialmente espressi presenti sul microarray. A questo punto, per ogni identificativo di entità incluso nell'esperimento, si ricavano le annotazioni ai termini di un vocabolario controllato scelto per l'analisi; si effettua quindi un test di ipotesi statistico per rilevare il livello di significatività (p-value) di ogni termine biomedico annotato nei due insiemi di riferimento: si tratta quindi di rilevare una sovra-rappresentazione (arricchimento) o sotto-rappresentazione (impoverimento) dei termini biomedici considerati nell'insieme *target* rispetto ai termini dell'insieme *master*. Sono anche possibili casi in cui non vi è un *master set*, ma due *target set*: essi possono essere disgiunti o sovrapposti (figura 2.6).

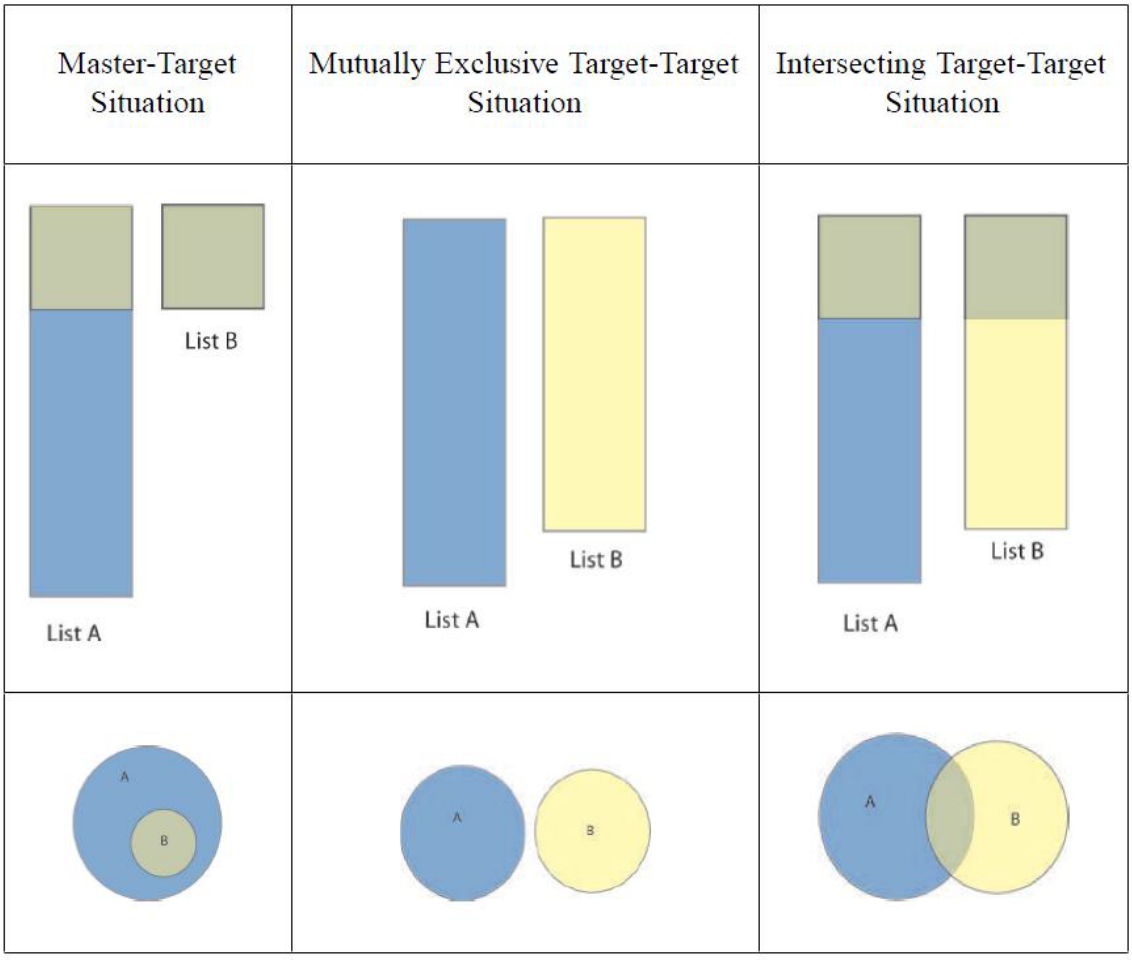


Figura 2.6. Possibili situazioni in analisi di arricchimento

2.4.1 Analisi statistica e correzione di test multipli

Il test d'ipotesi statistico comprende due ipotesi: quella nulla (indicata con *H0*) e quella alternativa (indicata con *H1*). Esso si usa per valutare la loro bontà: il metodo consente di formulare delle ipotesi, dedurne le conseguenze e valutare se la realtà effettivamente osservata si accorda con le deduzioni. Il procedimento è il seguente:

1. si fissano l'ipotesi nulla e l'ipotesi alternativa da testare e si valutano eventuali assunzioni statistiche per il test
2. si sceglie un test statistico appropriato da utilizzare
3. si ricava la distribuzione del test statistico sotto l'ipotesi nulla
4. si calcola il valore osservato della distribuzione del test statistico
5. si confronta il valore osservato con quello atteso, e in base a ciò si decide se rifiutare o accettare l'ipotesi nulla.

La conduzione di un test statistico comporta un rischio di errore, e in pratica ne esistono due tipi:

- rifiutare l'ipotesi nulla quando è vera determina un errore di primo tipo (indicato con α, tasso di falsi positivi)
- accettare l'ipotesi nulla quando è falsa determina un errore di secondo tipo (indicato con β, tasso di falsi negativi).

In particolare, nel caso dell'analisi di arricchimento [6]:

- sotto l'ipotesi nulla l'appartenenza di un'entità biomolecolare allo *study set* è indipendente dal fatto che essa sia annotata a un certo termine biomedico
- l'ipotesi alternativa prevede che le due condizioni sopra menzionate siano dipendenti.

Se il livello di significatività osservato è inferiore a quello atteso (p-value$\leq\alpha$) si rifiuta l'ipotesi nulla, cioè si giunge alla conclusione che l'appartenenza allo *study* set e l'annotazione al termine biomedico considerato siano dipendenti; se il livello di significatività è superiore si accetta l'ipotesi nulla e quindi l'indipendenza fra le due condizioni.

Per effettuare dei test d'ipotesi è possibile sfruttare diversi test statistici: nell'ambito dell'analisi di arricchimento di annotazioni biomolecolari i più utilizzati sono il test di Fisher, il test χ^2 di Pearson, il test sulla binomiale [7]. Per questa Tesi sarà utilizzato il test di Fisher.

Il test di Fisher [8] è un test per la verifica di ipotesi che si basa sulle tabelle di contingenza (cioè tabelle 2x2) che contengono le numerosità di oggetti classificati per due criteri qualitativi. La tabella 2.1 mostra una tabella di contingenza usata per il Test di Fisher.

	X0	X1	Totale Y
Y1	K-x	X	K
Y0	(N-K)-(M-x)	M-x	N-K
Totale X	N-M	M	N

Tabella 2.1. Tabella di contingenza 2x2

Le tabelle di contingenza considerano le numerosità di due o più categorie di elementi (se più di due è necessario utilizzare più tabelle 2x2); nel caso dell'analisi di arricchimento si considerano le numerosità delle entità biomolecolari appartenenti o meno allo *study set* (categorie Y1 e Y0) e annotate o meno a un certo termine biomedico considerato (categorie X1 e X0). N rappresenta quindi il numero totale di entità presenti nel population set; M il numero totale di entità del population set annotate a un certo termine di vocabolario controllato; K il totale di entità presenti nello study set; x indica il numero di entità presenti nello study set annotate al termine biomedico considerato. Bisogna quindi calcolare la probabilità che si produca una tabella contenente le numerosità ricavate dall'esperimento sotto l'ipotesi nulla. Il test di Fisher permette di calcolare il p-value su una o due code della distribuzione statistica. Si calcola il p-value su una coda tramite la formula

$$P_T = \frac{K!\,(N-K)!\,(N-M)!\,M!}{(K-x)!\,x!\,(M-x)!\,((N-K)-(M-x))!\,N!}$$

che indica la probabilità che si produca una certa tabella con i dati considerati; si considera quindi la tabella a disposizione, e si decrementa la cifra corrispondente a x finché non raggiunge lo 0, modificando le altre numerosità per mantenere gli stessi totali. A questo punto si calcola con la formula illustrata sopra la probabilità, e quindi

ricorsivamente si calcolano le probabilità delle tabelle ottenute incrementando x di nuovo fino al valore di partenza. Il p-value si otterrà quindi con la seguente sommatoria:

$$P_{1coda} = \sum_{i=0}^{j} P_{Ti}$$

dove j è il numero di tabelle di cui è stata calcolata la probabilità P_T.

La significatività dei p-value è legata alla probabilità di rifiutare l'ipotesi nulla erroneamente in un test singolo: se vengono effettuati molteplici test simultaneamente è possibile che la significatività di un p-value risulti aumentata di n volte (con n test effettuati); quindi è necessario effettuare una correzione ai p-value calcolati. Esistono diversi metodi per la correzione di test multipli; i più utilizzati sono:

- Bonferroni [9]: diminuisce molto i falsi positivi ma aumenta eccessivamente i falsi negativi (fa parte delle correzioni di tipo FWER, Family Wise Error Rate)
- Bonferroni-Holm [10]: anch'esso fa parte della categoria FWER, consiste in una correzione meno stringente all'aumentare dei valori dei p-value; aumentano i falsi positivi rispetto alla correzione Bonferroni, diminuiscono i falsi negativi
- Westfall-Young [11]: si basa su un certo numero di passi di resampling in cui avviene una permutazione delle entità biomolecolari presenti nello *study set*, vengono calcolati i p-value a ogni passo, e si ottiene il p-value corretto come il minimo fra i p-value calcolati che sono inferiori al valore originale. Poiché consiste in un certo numero di permutazioni da eseguire, il metodo è piuttosto lento [12]. Anch'esso fa parte della categoria FWER
- Benjamini-Hochberg [13]: esponente della categoria di correzioni FDR (False Discovery Rate), si basa sul confronto fra i p-value calcolati e la proporzione attesa di falsi positivi sul totale questo metodo consente di diminuire, rispetto ai precedenti, il numero dei falsi negativi (aumenta però il tasso di falsi positivi).

2.4.2 Stato dell'arte dei sistemi di analisi di arricchimento

Uno fra i più noti e utilizzati sistemi che offrono la possibilità di effettuare analisi di arricchimento di annotazioni biomolecolari è DAVID [14] (Database for Annotation, Visualization and Integrated Discovery). DAVID è un'applicazione web che consente il

caricamento di identificativi di geni derivanti da esperimenti biomolecolari e la loro annotazione e analisi (non solo di arricchimento). Le funzionalità principali [15] sono:

- Functional Annotation: effettua l'analisi di arricchimento di annotazioni e mostra l'appartenenza dei geni caricati a delle macro categorie di annotazioni
- Gene Functional Classification: fornisce un metodo per raggruppare le liste di geni caricate in gruppi funzionali, per comprendere meglio il contenuto biologico catturato da esperimenti con tecnologie high-throughput
- Gene ID Conversion: converte liste di ID di geni in formati diversi da quello inserito sfruttando un repository per il mapping.

Un altro strumento che permette di effettuare analisi di arricchimento e GFINDer (Genome Function INtegrated Discoverer), sviluppato dal Politecnico di Milano, che permette l'annotazione di liste numerose di geni con le informazioni biologiche fornite da cinque vocabolari controllati [16]. GFINDer permette di caricare liste di identificativi di geni con una classificazione per ogni ID, di annotare queste liste ai termini dei vocabolari controllati importati nel database sottostante (GO [17], PFAM [18], KEGG [3], OMIM [19]) e infine consente di effettuare un'analisi statistica delle annotazioni recuperate utilizzando un test statistico a scelta fra ipergeometrica, binomiale, Test Esatto di Fisher, Z-test e Poisson.

Uno strumento leggermente diverso è Ontologizer [20], che permette di eseguire analisi di arricchimento di annotazioni di identificativi di geni solamente sulla Gene Ontology. Ontologizer è un'applicazione Java utilizzabile e scaricabile liberamente; fra le funzionalità offerte vi è la possibilità di caricare diversi *study set* in corrispondenza di un unico *population set* in modo da poter effettuare più analisi simultaneamente. Per l'analisi viene utilizzato il Test di Fisher a una coda [21], e sono presenti i più diffusi metodi di correzione di test multipli. Un'altra utile funzionalità offerta è la possibilità di visualizzare i risultati dell'analisi direttamente sul grafo della Gene Ontology, evidenziando quindi i termini che hanno ottenuto p-value migliori.

2.5 Service Oriented Architecture (SOA)

L'espressione Service-Oriented Architecture sta generalmente ad indicare un'architettura software adatta a supportare l'uso di servizi web per garantire l'interoperabilità tra diversi sistemi così da consentire l'utilizzo delle singole applicazioni come componenti del processo di business e soddisfare le richieste degli

utenti in modo integrato e trasparente. Focalizzando l'attenzione sul concetto di servizio, gli attori in causa sono il fornitore e il richiedente del servizio. Questo tipo di interazione è il medesimo che si riscontra nei paradigmi client-server. Nella SOA, questa interazione viene arricchita con un terzo attore, il Service Broker, che si inserisce all'interno della comunicazione fra fornitore e fruitore del servizio [22].

Di seguito viene riportata una descrizione del ruolo dei tre attori coinvolti nella SOA (come illustrato in fig. 2.7):

- Service Provider: chi che realizza e mette a disposizione un servizio tramite l'operazione di *publish*, il servizio viene pubblicato, e le informazioni su esso vengono memorizzate in un registro accessibile pubblicamente
- Service Broker: si occupa della gestione del registro dei servizi consentendo di ricercare un servizio sulla base delle caratteristiche con le quali è stato memorizzato nel registro
- Service Requestor: colui che richiede un servizio. A tale scopo, l'utente interagisce con il Service Broker tramite la primitiva di *find* per ottenere il servizio ricercato. Una volta individuato il servizio e il suo fornitore, il richiedente si collega al Service Provider tramite la primitiva *bind* e può quindi fruire del servizio (utilizzando la primitiva *use*).

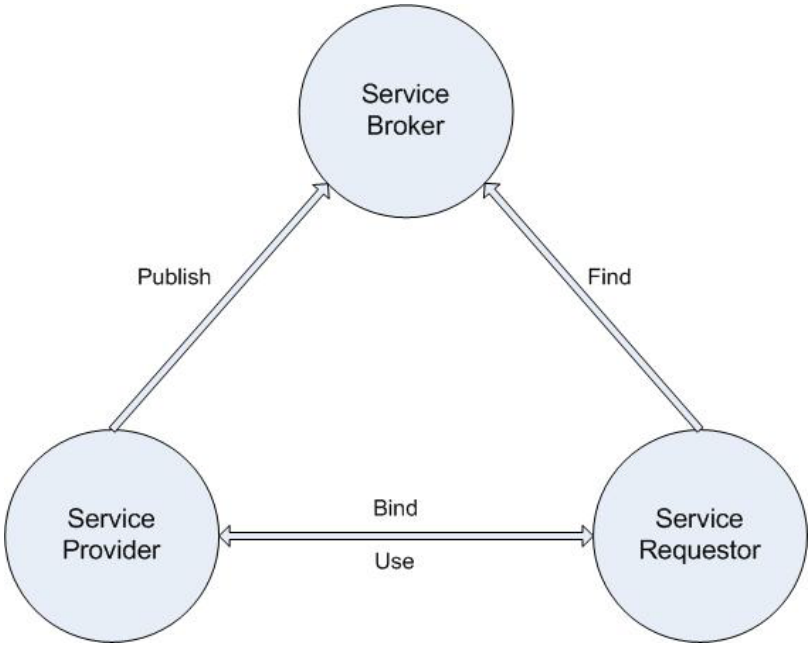


Figura 2.7. Schema dell'architettura SOA

I tre attori possono essere distribuiti fisicamente su macchine e piattaforme tecnologiche diverse; l'unico vincolo è l'utilizzo di un canale di trasmissione comune. Nell'ambito informatico, un'architettura basata su web service è un'istanza specifica di una SOA dove il canale di comunicazione considerato è il Web.

Le architetture SOA sono molto utilizzate in ambito bioinformatico: è possibile infatti creare dei workflow, processi più complessi che integrano le varie funzionalità dei servizi web sfruttati. Tali servizi vengono sviluppati da enti pubblici e privati, spesso da università e gruppi di ricerca, e vengono condivisi pubblicamente: ciò permette di ottenere dei processi distribuiti. Sfruttare tale architettura distribuita porta dei vantaggi sotto il punto di vista delle prestazioni: non è necessario avere a disposizione una grande potenza di calcolo per l'elaborazione dei dati biomolecolari, quindi le funzionalità di analisi di questi numerosi dati sono accessibili a chiunque possa connettersi a Internet.

Vi sono due paradigmi principali per le architetture di servizi web: il paradigma basato su SOAP, WSDL e UDDI e il paradigma REST.

2.5.1 SOAP

SOAP, WSDL e UDDI sono le tre tecnologie più utilizzate per i web service. WSDL (Web Service Description Language) è un linguaggio utilizzato per descrivere i servizi specificando la loro interfaccia e i metodi di invocazione. UDDI (Universal Description Discovery and Integration) è un registro (una base di date ordinata e indicizzata) basato su XML che permette la pubblicazione e la ricerca di servizi web; è progettato per essere interrogato da messaggi in formato SOAP e per fornire il collegamento ai documenti WSDL. SOAP [23] (Simple Object Access Protocol) è un protocollo per lo scambio di messaggi fra componenti software. Questo protocollo può operare su diversi protocolli di rete (il più utilizzato è HTTP) e racchiude i messaggi in un guscio (envelope) che include informazioni per effettuare la richiesta a un servizio web.

2.5.2 REST (Representational State Transfer)

REST è uno stile architetturale per la realizzazione di sistemi ipermediali distribuiti, ed è stato definito per la prima volta nella tesi di dottorato di Roy Fielding [24] (uno dei principali autori delle specifiche del protocollo HTTP). Secondo questo paradigma ogni informazione è considerata come una *risorsa* ed è accessibile tramite un identificatore (nel caso del Web un URL, in generale un URI). REST definisce anche una *rappresentazione* come lo stato corrente di una risorsa da trasferire fra i componenti della rete. REST è un'architettura di tipo client-server: i client fanno richieste di risorse ai server, che ne restituiscono una rappresentazione. La comunicazione (stateless) fra i componenti avviene tramite il protocollo HTTP usando i metodi standard GET, POST, PUT e DELETE. Secondo il paradigma REST client e server si scambiano rappresentazioni di una risorsa (ad esempio una pagina web) utilizzando uno di questi metodi: ad esempio la richiesta HTTP "GET www.polimi.it" permette di ottenere la pagina web che rappresenta la risorsa identificata dall'URL in questione.

I servizi web basati su REST (definiti RESTful) adottano questo semplice paradigma: ogni invocazione di servizio è stateless, non è necessario alcun protocollo aggiuntivo come SOAP e i vari servizi sono richiamabili tramite i classici metodi HTTP. Il richiedente del servizio necessita di conoscere semplicemente l'identificatore della risorsa (l'URI) e l'azione da effettuare; non è necessario conoscere eventuali componenti intermediari (come proxy, gateway, firewall) presenti fra il richiedente e il fornitore del servizio.

Capitolo 3

Obiettivi della tesi

Il Web è uno strumento indispensabile per la diffusione e la condivisione di conoscenza, e si moltiplicano ogni giorno le applicazioni e i servizi che offrono nuove e interessanti possibilità agli utenti. In particolare l'architettura basata su servizi offre delle possibilità impensabili fino a pochi anni fa: integrazione automatica di funzionalità in processi complessi, interoperabilità fra sistemi eterogenei e grande scalabilità. Le applicazioni e i servizi web, grazie al rapido sviluppo della tecnologia, utilizzano e generano un numero sempre maggiore di dati: tali dati devono quindi essere gestiti e memorizzati in modo efficiente, per garantire delle prestazioni di buon livello agli utenti della rete. I dati necessari a grandi sistemi informativi web frequentemente vengono integrati in basi di conoscenza, che permettono una maggiore facilità di raccolta, gestione, analisi e distribuzione delle informazioni contenute. Alcune di queste basi di conoscenza integrano dati già presenti nei database dell'organizzazione di appartenenza, altre integrano dati provenienti da sorgenti diverse ed eterogenee: la gestione risulta quindi più complessa e richiede maggiore attenzione. Se le sorgenti dati forniscono degli aggiornamenti dei dati, anche la base di conoscenza che li integra deve essere aggiornata conseguentemente: si parla in questi casi di basi di conoscenza in evoluzione. Un sistema informativo che si appoggi a una tale base di conoscenza deve essere in grado di gestire questa evoluzione. Inoltre, i sistemi informativi web che generano dati di utenti interagenti con la base di conoscenza devono integrare un meccanismo che permetta di mantenere la correlazione fra tali dati e la base di conoscenza durante la sua evoluzione. Se la quantità di dati generata è elevata, come avviene in sistemi data intensive, il processo per garantire il mantenimento di tale correlazione tra due successive versioni della base di conoscenza può richiedere un tempo lungo, durante il quale è tuttavia necessario mantenere l'operatività del sistema web e continuare a garantire il servizio offerto agli utenti. In tali casi non sono pertanto adeguati i meccanismi solitamente utilizzati per sistemi basati su data warehouse, i quali

prevedono l'aggiornamento off-line del data warehouse e il successivo veloce switch dalla versione precedente a quella aggiornata di tale data warehouse da parte dei sistemi che si appoggiano a esso.

I sistemi informativi web con le caratteristiche esposte sono utilizzati in molti campi; in particolare si è deciso di studiarne l'applicazione nell'ambito bioinformatico. In tale ambito, infatti, le problematiche sono amplificate dalla presenza di dati in quantità elevata provenienti da numerose sorgenti diverse offerte da enti pubblici e privati in tutto il mondo e aggiornati molto frequentemente (anche giornalmente). Inoltre, i sistemi bioinformatici spesso generano a loro volta grandi quantità di dati, in particolare durante l'analisi di dati sperimentali prodotti tramite l'uso di tecnologie high-throughput (come ad esempio i microarray). Pertanto, lo studio in tale ambito, che presenta criticità nei requisiti molto elevate, permetterà l'applicazione dei risultati anche in molti altri settori.

Obiettivo principale di questa Tesi è la progettazione e realizzazione di servizi e applicazioni web che si appoggiano a una base di conoscenza, che permettano una gestione efficiente dei numerosi dati in essa contenuti e dei dati generati durante l'esecuzione, e che siano in grado di gestire l'evoluzione della base di conoscenza minimizzando i tempi di inutilizzabilità da parte degli utenti.

Nello specifico caso bioinformatico considerato, la realizzazione di un sistema per l'analisi di arricchimento di annotazioni biomolecolari, si è deciso di sviluppare servizi web che si appoggiano alla Genomic and Proteomic Knowledge Base, una base di conoscenza realizzata nel Laboratorio di Basi di Dati del Dipartimento di Elettronica e Informazione del Politecnico di Milano che integra molti dati relativi a entità biomolecolari e feature biomediche provenienti dalle maggiori banche dati bioinformatiche mondiali. Attualmente tale base di conoscenza include circa 500 milioni di tuple e una volta al mese viene completamente rigenerata con i dati aggiornati delle banche dati considerate.

Per raggiungere questo scopo principale, gli obiettivi specifici di questa Tesi sono:

1. progettare una gestione efficiente dei dati provenienti da una base di conoscenza e dei numerosi dati generati dagli utenti di servizi o applicazioni che usano tale base di conoscenza
2. analizzare il problema del mantenimento dell'operatività di sistemi informativi web multi-utente durante l'evoluzione della base di conoscenza che utilizzano e

del mantenimento della correlazione fra i numerosi dati generati dagli utenti e quelli disponibili nella base di conoscenza, proponendone una soluzione

3. progettare e realizzare dei servizi web, in particolare per l'esecuzione di analisi bioinformatiche di arricchimento di annotazioni biomolecolari, che implementino la soluzione proposta per il problema dell'operatività di sistemi informativi durante l'evoluzione della base di conoscenza che questi utilizzano
4. progettare e realizzare un database di back-end per memorizzare i numerosi dati degli utenti di tali servizi web in modo efficiente in termini sia di spazio occupato, sia di tempo richiesto per l'interrogazione e l'estrazione di tali dati
5. sviluppare un'applicazione web di facile utilizzo che usi i servizi implementati e ne dimostri il funzionamento.

Capitolo 4

Definizione dei requisiti e scelte progettuali

Per raggiungere l'obiettivo principale esposto nel capitolo precedente, cioè realizzare servizi e applicazioni data intensive che sfruttino grandi basi di conoscenza in evoluzione in modo efficiente, vi sono alcuni requisiti generali principali che devono essere soddisfatti. Ovvero, è necessario:

- progettare dei servizi web che possano essere integrati in processi più complessi e che possano gestire grandi quantità di dati efficientemente
- progettare dei metodi per l'estrazione di dati da una grande base di conoscenza variabile nel tempo (sia per quanto riguarda i dati, sia per quanto riguarda lo schema dei dati)
- progettare un meccanismo per garantire, durante l'evoluzione della base di conoscenza, massima operatività ai servizi e alle applicazioni che la utilizzano e per mantenere la correlazione fra i potenzialmente numerosi dati generati dagli utenti e quelli della base di conoscenza.

Per quanto riguarda lo specifico caso preso in considerazione nello sviluppo della Tesi, quello bioinformatico, in cui i dati sono numerosi e sempre in evoluzione, è necessario costruire un sistema che consenta di supportare gli aggiornamenti e l'integrazione di dati provenienti da numerose sorgenti eterogenee e che sia di facile utilizzo da parte degli utenti. In particolare questa Tesi considera come caso d'esempio la realizzazione di un sistema di analisi di arricchimento di annotazioni biomolecolari utilizzabile sia via web service sia come applicazione web, che si appoggi a una grande base di conoscenza in evoluzione, la Genomic and Proteomic Knowledge Base (GPKB) integrante informazioni provenienti dalle più importanti banche dati biomolecolari e biomediche

mondiali, e che potenzialmente genera numerosi dati risultato delle analisi realizzate da ciascun utente.

Il sistema in oggetto deve quindi soddisfare alcuni requisiti specifici:

- fornire un meccanismo di robustezza agli aggiornamenti della base di conoscenza in modo che il sistema sviluppato sia operativo durante tali aggiornamenti e che i dati degli utenti siano sempre correlati a quelli della base di conoscenza
- consentire agli utenti di effettuare l'upload di liste di numerosi identificativi (ID) di entità biomolecolari, eventualmente dotati di dati di classificazione, solitamente derivanti da risultati di analisi sperimentali, genomici o proteomici, a elevata produzione di dati, eventualmente basate su microarray
- identificare ed eliminare errori grossolani nelle liste caricate dall'utente (ID duplicati, ID non conformi al formato della sorgente dati di appartenenza, etc.) e segnalarli all'utente
- effettuare la traduzione degli ID caricati nel formato della base di conoscenza GPKB, identificando e segnalando all'utente eventuali ID obsoleti o non riconosciuti
- effettuare l'annotazione degli ID caricati ai termini (di ontologie o terminologie) che descrivono le caratteristiche (feature) biomediche selezionate dagli utenti, con la possibilità di selezionare, per ogni caratteristica, la sorgente dati e il provider delle annotazioni biomolecolari da considerare
- effettuare simultaneamente l'analisi di arricchimento di tutte le caratteristiche selezionate dall'utente, basandosi sulle annotazioni recuperate dalla base di conoscenza, utilizzando (in funzione delle scelte dell'utente) uno fra i diversi test messi a disposizione ed eventualmente una delle correzioni per test multipli disponibili
- effettuare le analisi in modo efficiente e rapido, anche grazie all'utilizzo di librerie software che implementino gli algoritmi per l'analisi di arricchimento e per correzioni di test multipli in modo efficiente
- effettuare analisi di tipo sia genomico, sia proteomico, garantendo un'elevata qualità dei risultati, grazie all'efficace utilizzo della base di conoscenza integrata
- rendere sempre disponibili all'utente le versioni più aggiornate della base di conoscenza contenente i dati relativi a entità biomolecolari, caratteristiche biomediche e annotazioni

- rendere il sistema facilmente utilizzabile anche da utenti che non possiedono un'approfondita conoscenza informatica
- ridurre al minimo i tempi di risposta necessari per il caricamento di liste di ID (anche numerosi), per la traduzione di tali ID e per la loro analisi di arricchimento
- gestire utenti con diversi privilegi di accesso al sistema: utenti standard, avanzati e amministratore
- permettere a utenti avanzati di salvare le liste di ID caricate, le impostazioni e i risultati delle analisi
- consentire a utenti avanzati di riutilizzare proprie liste di ID o impostazioni salvate per effettuare nuove analisi.

Dati tali requisiti sia generali che specifici, si è deciso di realizzare un sistema per effettuare analisi di arricchimento di annotazioni di entità biomolecolari come segue: l'utente può caricare una o più liste di ID di entità biomolecolari, e selezionare eventualmente dei sottoinsiemi di ID in base a valori di classificazione a essi associati; deve poi selezionare il tipo di analisi di arricchimento da effettuare, il test statistico e la correzione per test multipli.
Riguardo alla modalità di realizzazione dell'analisi di arricchimento, normalmente tali analisi vengono effettuate basandosi sui dati associati agli ID forniti da una certa banca dati. Nell'analisi si è quindi limitati ai dati forniti da tale sorgente e ciò non permette di recuperare ulteriori informazioni relative alla stessa entità biologica o alla stessa caratteristica biomedica provenienti da altre fonti. Si è pertanto deciso di realizzare anche un'analisi concettuale. Per analisi concettuale si intende la possibilità di utilizzare nell'analisi di arricchimento tutte le informazioni disponibili sulle entità biologiche e caratteristiche biomediche in esame e non solo quelle associate a uno specifico identificativo fornite da una specifica sorgente dati; ciò è possibile riunendo tutte le informazioni relative a diversi identificativi provenienti da diverse fonti tramite dati di similarità fra entità biomolecolari e caratteristiche biomediche integrati nella base di conoscenza. L'analisi concettuale permette quindi di sfruttare una conoscenza più completa sulle entità coinvolte nell'analisi, e quindi di ottenere risultati più completi e significativi.
Nel sistema sviluppato durante questa Tesi si è deciso di implementare e rendere disponibili quattro diversi tipi di analisi:

- analisi basata solo sugli specifici ID disponibili nelle sorgenti dati: vengono recuperate le annotazioni fra gli ID caricati dall'utente e gli ID dei termini delle feature biomediche selezionate per l'analisi; in questo tipo di analisi non si tiene conto di un'eventuale corrispondenza fra più ID inseriti dall'utente rappresentanti la stessa entità, né si tiene conto di similarità fra ID di termini di feature biomediche provenienti da sorgenti diverse
- analisi concettuale lato entità biomolecolare: vengono recuperate tutte le annotazioni fra l'entità biomolecolare corrispondente all'ID inserito dall'utente e gli ID dei termini delle feature biomediche selezionate per l'analisi; non si tiene conto di similarità fra ID di feature biomediche di sorgenti diverse, ma si considerano tutte le annotazioni dell'entità biomolecolare a tali feature biomediche, non solo quelle disponibili per i soli ID dell'entità biomolecolare inseriti dall'utente
- analisi concettuale lato feature biomedica: vengono recuperate le annotazioni fra gli ID caricati dall'utente e tutti i termini delle feature biomediche selezionate per l'analisi, considerando quindi per i soli ID caricati dall'utente tutte le annotazioni provenienti da qualunque sorgente dati per le feature biomediche selezionate
- analisi concettuale sia a livello di entità biomolecolare, sia a livello di feature biomedica: vengono estratte e analizzate tutte le annotazioni fra ogni entità biomolecolare corrispondente agli ID inseriti dall'utente e ogni feature biomedica, indipendentemente dagli specifici ID caricati dall'utente e dalle loro specifiche annotazioni a specifici termini delle feature biomediche selezionate per l'analisi fornite da particolari sorgenti dati.

Nel caso di analisi concettuale lato feature biomedica per feature biomediche espresse mediante ontologie, è necessario considerare un aspetto fondamentale; poiché l'analisi viene effettuata su annotazioni fornite da più sorgenti diverse per ogni data feature, in generale tali annotazioni sono relative a più ontologie diverse; ogni sorgente dati fornisce infatti un'ontologia per una data feature, quindi più sorgenti portano ad avere più ontologie per la stessa feature. Per non introdurre errori nell'analisi è quindi necessario effettuare un passaggio preliminare di integrazione delle varie ontologie in una unica rappresentante la feature considerata.

Riguardo all'implementazione delle varie funzionalità del sistema progettato, si è scelto di realizzare servizi separati per il caricamento, controllo e traduzione delle liste di ID e

per l'analisi di arricchimento vera e propria; tali servizi sono richiamati in sequenza per realizzare l'analisi completa.

Capitolo 5

Materiali e metodi

5.1 Service oriented architecture

5.1.1 RESTful Web Services

In questa Tesi si è deciso di seguire il paradigma REST per lo sviluppo di servizi web.
I servizi web di tipo RESTful si basano sui principi esposti nel capitolo 2.5.2: ogni servizio sviluppato in questa Tesi viene trattato come una risorsa che possiede un identificativo univoco (l'URL a cui raggiungerla) ed è invocabile tramite uno dei quattro metodi standard HTTP. Per i web service basati su REST, non è necessario esporre l'interfaccia in un file descrittore; è comunque possibile farlo utilizzando uno dei due linguaggi disponibili: WSDL [25] (che supporta servizi RESTful a partire dalla versione 2.0) o WADL [26] (Web Application Description Language), che fornisce una rappresentazione di tipo XML di una qualsiasi applicazione web basata su HTTP.
I web service basati su REST sono equivalenti, in fatto di funzionalità, ai servizi basati su SOAP. I servizi REST, essendo basati semplicemente su richieste HTTP, possono essere utilizzati e integrati semplicemente i processi complessi senza richiedere l'uso di protocolli aggiuntivi come invece avviene per servizi basati su SOAP.

5.1.2 Apache CXF e JAX-RS

Per lo sviluppo dei web service è stato utilizzato il frame work di sviluppo Apache CXF (27) e in particolare la sua implementazione delle specifiche JAX-RS (Java API for RESTful Web Services)[28]. JAX-RS semplifica lo sviluppo di web service RESTful grazie all'utilizzo di annotazioni per il mapping fra risorse e classi Java. In particolare le principali annotazioni sono:

- @GET, @POST, @PUT, @DELETE mappano le richieste HTTP sulle funzioni della classe che implementa il metodo corrispondente

- @Path specifica il percorso relativo da inserire nell'URL della richiesta per raggiungere la risorsa
- @Produces, @Consumes indicano i tipi di output ed input della richiesta.
- @Query Param specifica il mapping fra un parametro della richiesta HTTP e un parametro della funzione corrispondente
- @PathParam specifica il mapping fra una variabile di percorso (cioè una parte dell'indirizzo URL) e un parametro di un metodo Java.

Per il passaggio di oggetti fra client e server è necessario effettuare la loro serializzazione: CXF include le librerie JAXB (Java Architecture for XML Binding) per effettuare il mapping fra classi Java e la loro rappresentazione XML. Anche JAXB utilizza delle annotazioni, le principali sono:

- @XMLRootElement effettua il mapping fra una classe Java e la radice del file XML
- @XMLAttribute effettua il mapping fra un attributo di una classe Java e un attributo di un elemento XML
- @XMLElement effettua il mapping fra un attributo di una casse Java e un elemento XML.

Utilizzando le annotazioni JAXB è possibile sfruttare la serializzazione (e deserializzazione) automatica delle risorse evitando di dover implementare dei parser per la gestione dei dati provenienti dal servizio web (in qualunque formato essi siano). Utilizzando JAXB si hanno comunque a disposizione diversi formati di output: XML, JSON, HTML o semplice testo.

5.2 Web Server: Apache Tomcat

Per la pubblicazione dell'applicazione e dei servizi web si è scelto di utilizzare Apache Tomcat [29], uno dei più diffusi web server comprendente un servlet container, che permette quindi l'esecuzione sia di servizi sia di servlet Java.

5.3 Strumenti per la gestione e l'elaborazione dei dati

5.3.1 PostgreSQL

Il DBMS (DataBase Management System) utilizzato per la gestione dei dati è PostgreSQL [30]. PostgreSQL è un DBMS relazionale e ad oggetti di tipo open source e sviluppato da una comunità. La versione utilizzata è la 8.4, e supporta tutte le principali caratteristiche e funzionalità del linguaggio SQL; in particolare per l'applicazione sviluppata in questa Tesi vengono sfruttate due caratteristiche avanzate: il partizionamento di tabelle e l'utilizzo di stored procedures. Le stored procedures verranno trattate più dettagliatamente nel capitolo 5.3.2. Per quanto riguarda il partizionamento, PostgreSQL lo supporta in maniera diversa dalla maggior parte dei DBMS esistenti: infatti le partizioni non sono vere e proprie tabelle fisiche, ma sono anch'esse normali tabelle e il partizionamento viene ottenuto creando un albero di ereditarietà fra esse; così le tabelle figlie contengono fisicamente i dati, e dalla tabella padre (che verrà trattata come l'equivalente di una tabella logica) è possibile recuperare tutti i dati dei figli. PostgreSQL permette anche l'inserimento di dati nella tabella padre; questo però si discosta dalla definizione di partizionamento, quindi in questa Tesi verrà trattato solo il caso in cui le tabelle padre non contengono dati propri. La scelta del DBMS è ricaduta su PosgreSQL poiché a esso si appoggia la base di conoscenza (GPKB) utilizzata per l'estrazione dei dati biomolecolari.

5.3.2 pl/pgSQL

Le stored procedures sono dei programmi che possono essere eseguiti direttamente all'interno del database. PosgreSQL supporta la scrittura di stored procedures in molti linguaggi diversi, come ad esempio PL/pgSQL, PL/Tcl, PL/Perl, PL/Java e altri ancora. Per questa Tesi è stato utilizzato PL/pgSQL, linguaggio già integrato in PostgreSQL, che permette di eseguire tutti i comandi SQL e mette a disposizione i principali costrutti dei classici linguaggi di programmazione (costrutti condizionali, cicli, eccezioni) e un gran numero di funzioni di utilità già presenti in PostgreSQL. L'utilizzo di stored procedures è molto vantaggioso in termini di prestazioni: infatti, l'esecuzione di grandi quantità di comandi SQL da applicazioni esterne causa degli overhead non indifferenti;

sfruttando queste procedure, invece, i comandi vengono eseguiti all'interno del database e l'esecuzione risulta più veloce [31].

5.3.3 Java

Per lo sviluppo del codice si è scelto di utilizzare Java [32], linguaggio molto versatile e che garantisce elevata portabilità. Java, inoltre, consente una facile integrazione sia con le tecnologie web (come ad esempio il framework CXF), sia con i DBMS.

5.3.4 JDBC

Per accedere al database dall'applicazione è necessario utilizzare un driver per la connessione. La scelta obbligata è stata JDBC (Java DataBase Connectivity) [33], driver appositamente sviluppato per Java. In particolare è stata utilizzata la versione implementata specificamente per PostgreSQL. Tramite il driver è possibile effettuare tutte le operazioni SQL di base come connessioni al database, query di estrazione dati, operazioni di manipolazione dei dati, istruzioni DDL e anche operazioni più avanzate come l'esecuzione di operazioni in batch.

5.3.5 JDOM

Per la gestione dei file di configurazione dell'applicazione si è scelto di utilizzare il linguaggio XML (eXtensible Markup Language) linguaggio basato su tag per la definizione di dati. Per gestire i file XML da Java si è scelto di usare la libreria JDOM [34], che implementa l'interfaccia DOM (Document Object Model)[35] per la manipolazione di XML. Tramite JDOM viene effettuato il parsing del documento XML e viene generato l'albero corrispondente per una facile gestione degli elementi contenuti.

5.4 Strumenti per la presentazione dei dati

Per la presentazione di contenuti all'utente è stato necessario utilizzare diverse tecnologie: per i contenuti statici HTML, per quelli dinamici JSP e JavaScript.

5.4.1 Lato server: JSP e Servlet

Per effettuare richieste ai web service sono state utilizzate delle servlet Java. Le servlet sono classi che fanno parte della piattaforma di sviluppo Java EE (Enterprise Edition) [36]; sono delle speciali classi Java che vengono eseguite su un web server che abbia anche la funzione di servlet container (in questo caso Tomcat). Le servlet sono classi che vengono caricate dal web server all'avvio, e successivamente ricevono richieste HTTP da parte di pagine web o applicazioni; esse sono persistenti, cioè rimangono attive per più richieste. Hanno tre metodi principali:

- init(): serve a inizializzare la servlet
- service(): è il metodo utilizzato per rispondere alle richieste HTTP (e comprende i metodi doGet(), doPost(), doPut(), doDelete()
- destroy(): metodo per la terminazione della servlet.

Per generare contenuti dinamici nelle pagine web dell'applicazione le servlet sono state utilizzate insieme alla tecnologia JSP (Java Server Pages). Le JSP sono semplici pagine HTML contenenti dei frammenti di codice Java che consentono l'inserimento di dati dinamici in una pagina web. Le principali variabili a disposizione in una pagina JSP sono:

- request: rappresenta la richiesta HTTP
- response: rappresenta la risposta HTTP
- session: rappresenta la sessione HTTP all'interno della quale è stata invocata la pagina.

5.4.2 Scripting dinamico: AJAX

Per garantire una maggiore usabilità all'applicazione si è deciso di inserire dei contenuti dinamicamente anche a pagina web già caricata: per questo è stato necessario l'utilizzo del linguaggio di scripting Javascript e di AJAX [37] (Asynchronous JavaScript and XML), una tecnica per eseguire richieste HTTP al server in maniera asincrona. AJAX si basa sull'oggetto XMLHttpRequest per lo scambio di dati asincrono con il web server; nonostante sia menzionato XML, può essere utilizzato qualunque formato per lo scambio di dati. AJAX ha il vantaggio di poter generare più contenuti dinamicamente all'interno della stessa pagina web, ma ha lo svantaggio di richiedere l'attivazione di JavaScript sul client e di non essere compatibile con i browser meno recenti.

5.4.3 Lato client: HTML e CSS

Per la presentazione dei dati all'utente la scelta obbligata nell'ambito web è il linguaggio HTML (HyperText Markup Language). Questo linguaggio permette di presentare i contenuti dell'applicazione formattati secondo uno stile definito in un file CSS (Cascading Style Sheets), che permette di gestire il posizionamento degli elementi nella pagina web.

5.5 Annotazioni biomolecolari e strumenti per la loro analisi

Per sviluppare il software per analisi di arricchimento trattato in questa Tesi, sono stati utilizzati due strumenti principali a supporto: la Genomic and Proteomic Knowledge Base, una base di conoscenza integrante informazioni relative a entità biomolecolari, feature biomediche e annotazioni, e Ontologizer, software open source che implementa degli algoritmi per l'analisi statistica di annotazioni.

5.5.1 Genomic and Proteomic Knowledge Base (GPKB)

La Genomic and Proteomic Knowledge Base (GPKB) è una base di conoscenza sviluppata al Laboratorio di Basi di Dati del Dipartimento di Elettronica e Informazione del Politecnico di Milano per l'importazione e l'integrazione di dati provenienti da tutte le maggiori banche di dati biomolecolari e biomedici ed è contenuta nel Genomic and Proteomic Data Warehouse (GPDW) [38]. La base di conoscenza è strutturata su tre livelli principali: il livello di importazione, nel quale sono importati nelle tabelle tutti i dati scaricati dalle principali banche dati mondiali; il livello di integrazione, nel quale questi dati sono integrati in tabelle comuni per ogni entità considerata; il terzo livello, ancora in sviluppo, nel quale i dati sono integrati a livello concettuale. L'integrazione a livello concettuale implica che diversi ID corrispondenti alla stessa entità biologica o alla stessa feature biomedica vengano identificati con un codice unico (chiamato Concept OID) all'interno della base di conoscenza: grazie a ciò è possibile riunire sotto

un solo identificativo tutte le informazioni nella base di conoscenza relative a una certa entità biomolecolare o biomedica ed ottenere una conoscenza più completa. Poiché la base di conoscenza è in continua evoluzione e subisce aggiornamenti molto frequenti, per sfruttarla è necessario appoggiarsi alle tabelle di metadati integrate in essa: estraendo i metadati è infatti possibile ottenere informazioni sullo schema effettivo dei dati ed effettuare query dinamicamente. Infatti lo schema del GPDW è soggetto a frequenti variazioni necessarie per conformarsi a eventuali cambiamenti dei requisiti delle sorgenti dati (la base di conoscenza viene rigenerata ogni mese per integrare gli aggiornamenti forniti dalle banche dati a cui si apppoggia). Il GPDW include i dati di tutte le entità biomolecolari (geni, proteine, sequenze nucleotidiche, sequenze amminoacidiche), dei principali vocabolari controllati (ontologie e terminologie) e dati di annotazioni biomolecolari e di similarità; le dimensioni del data warehouse sono molto elevate, circa 250 GB, e il numero di tuple contenute si aggira in totale sui 500 milioni. Una delle peculiarità della GPKB è la possibilità di offrire annotazioni dirette non solo per i geni, ma anche per le proteine. A differenza della maggior parte degli strumenti disponibili sul Web, che offrono annotazioni proteomiche indirette ricavate traducendo preliminarmente gli identificativi di proteine in identificativi di geni, e in seguito ricercando le annotazioni associate a questi ultimi, la GPKB contiene informazioni sulle annotazioni esplicitamente associate alle proteine.

5.5.2 Ontologizer

Ontologizer [20] è un tool per l'analisi di arricchimento di annotazioni di liste di geni espresse mediante la Gene Ontology [17]. I metodi utilizzati, però, sono adattabili senza sforzo a un utilizzo con altre ontologie o terminologie, e all'analisi di liste di proteine o di qualunque altra entità biomolecolare. Tutto il codice di Ontologizer è open source ed è scaricabile tramite un programma di SVN.

Ontologizer offre un'interfaccia grafica di tipo Swing di Java, e per ottenere i dati delle ontologie e delle annotazioni esegue un parsing su flat file forniti dalla Gene Ontology. Poiché per questa tesi verranno implementati servizi web e i dati saranno recuperati dalla Genomic and Proteomic Knowledge Base, i due moduli per la gestione di interfaccia grafica e parsing dei file di Ontologizer verranno tralasciati nella descrizione. Ontologizer è in grado di costruire i grafi delle ontologie recuperate da flat file scaricati dalle banche dati e di ricavare le annotazioni dirette associate a ogni termine

(recuperando anch'esse da flat file di annotazioni); inoltre è in grado di ricavare anche le annotazioni indirette tramite l'attraversamento della struttura del grafo generato [39].
Sono presenti diversi metodi per il calcolo dei p-value per l'analisi di arricchimento di annotazioni ai termini dei vocabolari controllati e altrettanti metodi per la correzione di test multipli.
Per il calcolo dei p-value Ontologizer si basa sul Test esatto di Fisher (nella versione a una coda), implementato nella classe ontologizer/Hypergeometric.java. Tutti gli algoritmi per l'analisi di arricchimento si basano su questa classe. In particolare sono stati implementati degli algoritmi specifici per ontologie che effettuano delle correzioni ai p-value tenendo conto della dipendenza fra termini del grafo.
Gli algoritmi disponibili (implementati nel package 'calculation') sono:

- Term-for-term [21] che considera tutti i termini di vocabolari controllati indipendenti fra loro (applicabile indifferentemente a terminologie e ontologie). Questo algoritmo realizza in pratica un'analisi di arricchimento standard basata sul Test di Fisher
- Parent-Child Union e Parent-Child-Intersection [40] (solo per ontologie), che sono implementazioni del metodo di Grossman (che considerano rispettivamente i geni della population set o dello study set annotati all'insieme di antenati di un termine come l'unione o l'intersezione dei geni annotati a ogni singolo antenato)
- Topology-weighted e Topology-elim [41] (solo per ontologie), che sono implementazioni del metodo di Alexa (con eliminazione o pesatura dei termini correlati a un dato termine)
- MGSA [42], metodo basato su reti bayesiane, per come è implementato non sfrutta la struttura del grafo dell'ontologia, è perciò utilizzabile anche per terminologie. Su questo metodo non è necessario applicare una correzione per test multipli.

I metodi di multiple test correction disponibili sono:

- Bonferroni [9]
- Bonferroni-Holm [10]
- Benjamini-Hochberg (del tipo FDR) [13]
- Benjamini-Yekutieli (del tipo FDR) [43]
- Westfall-Young [11] single-step (a un passo)
- Westfall-Young step-down (a n passi).

Per quanto riguarda l'implementazione specifica di Ontologizer, è realizzata in questo modo: esiste un'interfaccia ICalculation, che viene chiamata dalla classe che desidera effettuare il calcolo dei p-value. Tramite un metodo getCorrectionByName presente nella classe CalculationRegistry (un registro per tutti i metodi), viene istanziata la specifica implementazione dell'interfaccia. Si supponga per comodità di utilizzare la classe TermForTermCalculation come esempio, che esegua un calcolo presumendo i termini (dell'ontologia o della terminologia) indipendenti. Successivamente, richiamando i metodi di ICalculation verranno eseguiti quelli della specifica implementazione scelta (nel caso considerato in esempio, quelli di TermForTermCalculation). All'interno di TermForTermCalculation quindi verrà effettivamente eseguito il calcolo. Successivamente, sempre all'interno di TermForTermCalculation, verranno utilizzati i metodi per la correzione di test multipli richiamabili tramite la classe astratta AbstractTestCorrection. Come per il calcolo, il metodo specifico da utilizzare viene recuperato tramite un registro, TestCorrectionRegistry, e verranno quindi richiamati i metodi dell'implementazione specifica scelta (ad esempio Bonferroni-Holm, che è una classe che estende AbstractTestCorrection). È quindi la classe che si occupa del calcolo dei p-value, in questo caso TermForTermCalculation, a fornire il risultato finale (già completo di correzione) in un oggetto di tipo EnrichedGOTermResult.

Gli input necessari agli algoritmi per l'analisi di arricchimento sono: il grafo rappresentante l'ontologia (o un grafo fittizio rappresentante una terminologia), le liste di ID contenuti in population set e study set e le annotazioni dirette di termini biomedici sugli elementi di queste liste, e infine il metodo per la correzione di test multipli selezionato.

Vi sono altre classi principali da descrivere. La classe GOGraph crea il grafo dell'ontologia a partire dai dati sui termini estratti da un file .obo (flat file contenente informazioni su termini e relazioni dell'ontologia fornito dalla Gene Ontology). La classe Association rappresenta un'annotazione fra un gene (o entità biomolecolare) e un termine biomedico, AssociationContainer è un container di tutte le annotazioni. La classe Gene2Associations contiene tutte le annotazioni di un dato gene (o entità biomolecolare) con i termini dell'ontologia (o terminologia) selezionata. La classe Term rappresenta un termine dell'ontologia (o terminologia), TermID e TermRelation sono due tipi di dati che rappresentano gli ID e i tipi di relazione esistenti. Le classi StudySet e PopulationSet, infine, rappresentano le liste di geni (o entità biomolecolari in genere)

caricate dall'utente, e contengono diversi metodi che agiscono su tali liste. Vi sono molte altre classi, ma poco significative oppure non utilizzate nello svolgimento di questa Tesi, e non verranno quindi descritte in dettaglio.

Capitolo 6

Risultati e discussione

6.1 Architettura del sistema realizzato

Per lo sviluppo del sistema si è scelto di utilizzare una basilare architettura a tre livelli (illustrata in fig. 6.1), che rispetta il paradigma MVC (Model-View-Controller); il livello di dati consiste di un database server che ospita il Genomic and Proteomic Data Warehouse e un database contenente i dati dell'applicazione e dei servizi sviluppati; il livello applicativo consiste di un web server Tomcat che ospita ed espone i servizi web; a livello di presentazione si considera l'interfaccia web utilizzabile dall'utente tramite browser. L'esposizione sia di servizi sia di un'applicazione web permette che il sistema venga utilizzato sia da utenti esperti, che potranno sfruttare e integrare i servizi web in processi più complessi, sia da utenti senza una preparazione informatica avanzata grazie alla facile e intuitiva interfaccia web. I client non hanno necessità di scaricare alcun software e tutto il carico di elaborazione viene sostenuto dai server: in questo modo il sistema sarà utilizzabile senza la necessità di particolari requisiti hardware. Il sistema software sfrutta il GPDW per estrarre dati di tipo biomolecolare e biomedico; il database ApplicationDB viene utilizzato per memorizzare i dati stabili degli utenti; il sistema informativo sviluppato possiede anche uno schema di dati all'interno del GPDW, contenente i dati che interagiscono con la base di conoscenza. Il database GPDW System offre invece informazioni sulle versioni della base di conoscenza e sui suoi aggiornamenti.

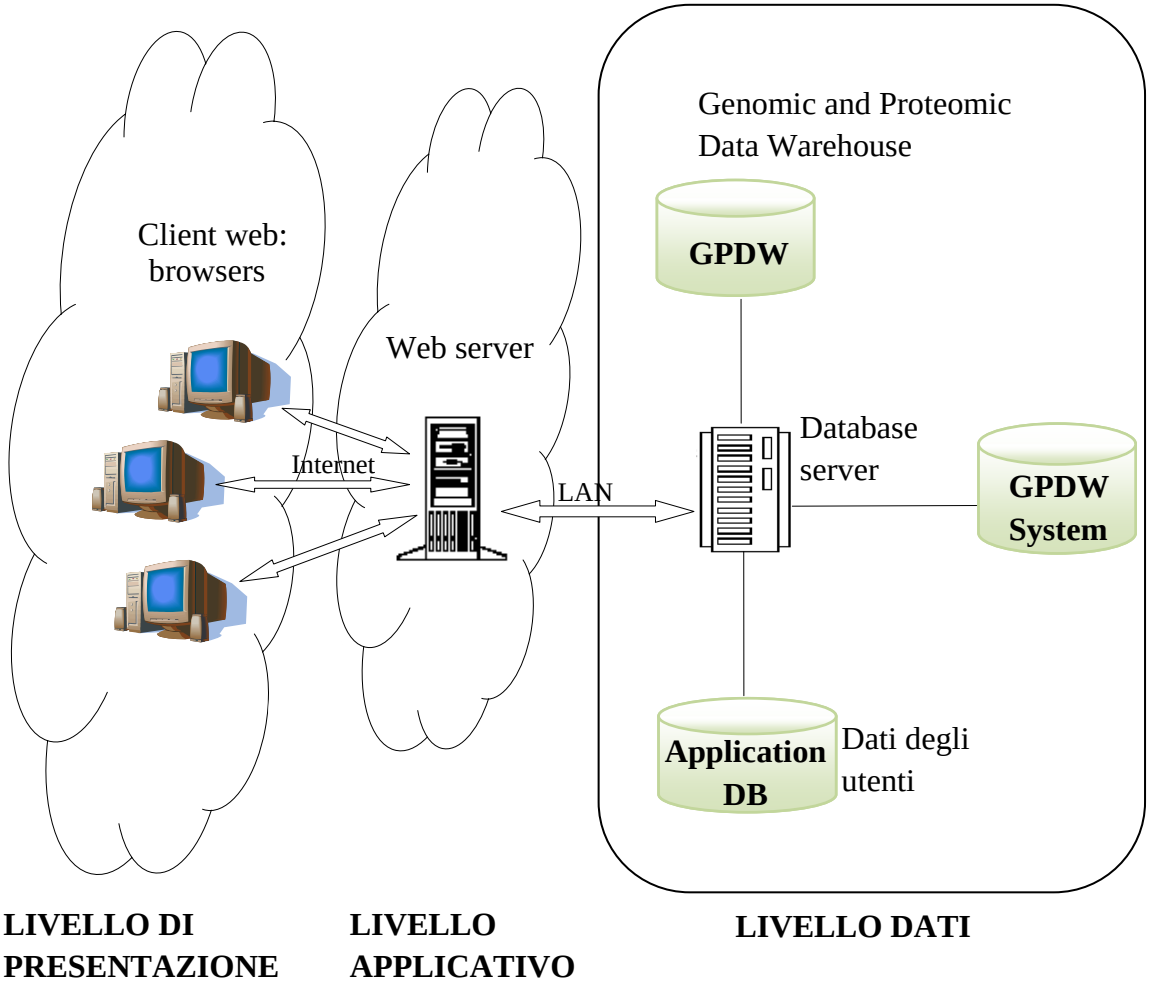


Figura 6.4. Architettura a tre livelli del sistema.

In fig. 6.2 sono illustrati i casi d'uso del sistema. Sono presenti quattro tipologie di utenti: l'utente anonimo può solamente registrarsi, mentre gli utenti standard e avanzati possono compiere un certo numero di operazioni (l'utente avanzato dispone, rispetto all'utente standard, anche delle operazioni di salvataggio e riutilizzo dei dati). L'amministratore ha pieno potere sul sistema e può eseguire tutte le operazioni eseguite da un utente avanzato (ciò non è indicato nel diagramma per facilità di lettura); inoltre ha il potere di gestire tutti i dati presenti nel sistema. Le principali operazioni disponibili sono il caricamento di liste di ID di entità biomolecolari, la definizione di un esperimento e di tutte le sue impostazioni, l'esecuzione dell'analisi. L'annotazione è

un'operazione interna al sistema necessaria per l'analisi di arricchimento, ma non è una funzionalità offerta agli utenti. Tutte le operazioni richiedono una fase di login.

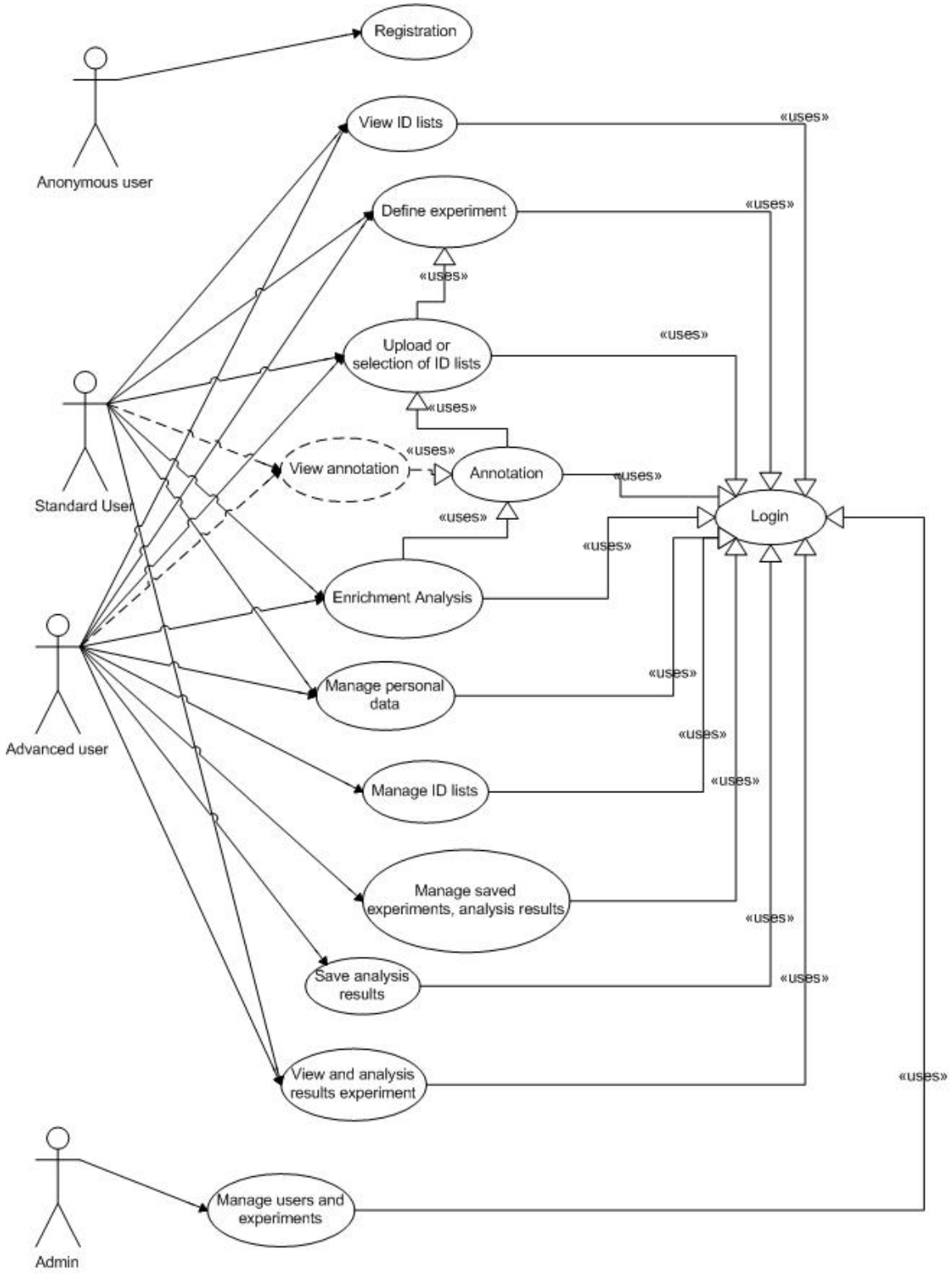


Figura 6.2. Diagramma dei casi d'uso del sistema

In fig.6.3 è invece illustrato lo schema dei vari moduli software che compongono il sistema realizzato. Essi si possono dividere in alcuni gruppi principali: il gruppo di moduli per la gestione e l'estrazione di dati, il gruppo di moduli di elaborazione, i moduli per l'esposizione dei servizi web e i moduli che implementano l'applicazione web (i vari gruppi sono divisi per colore nello schema; il significato di ogni colore è indicato in legenda).

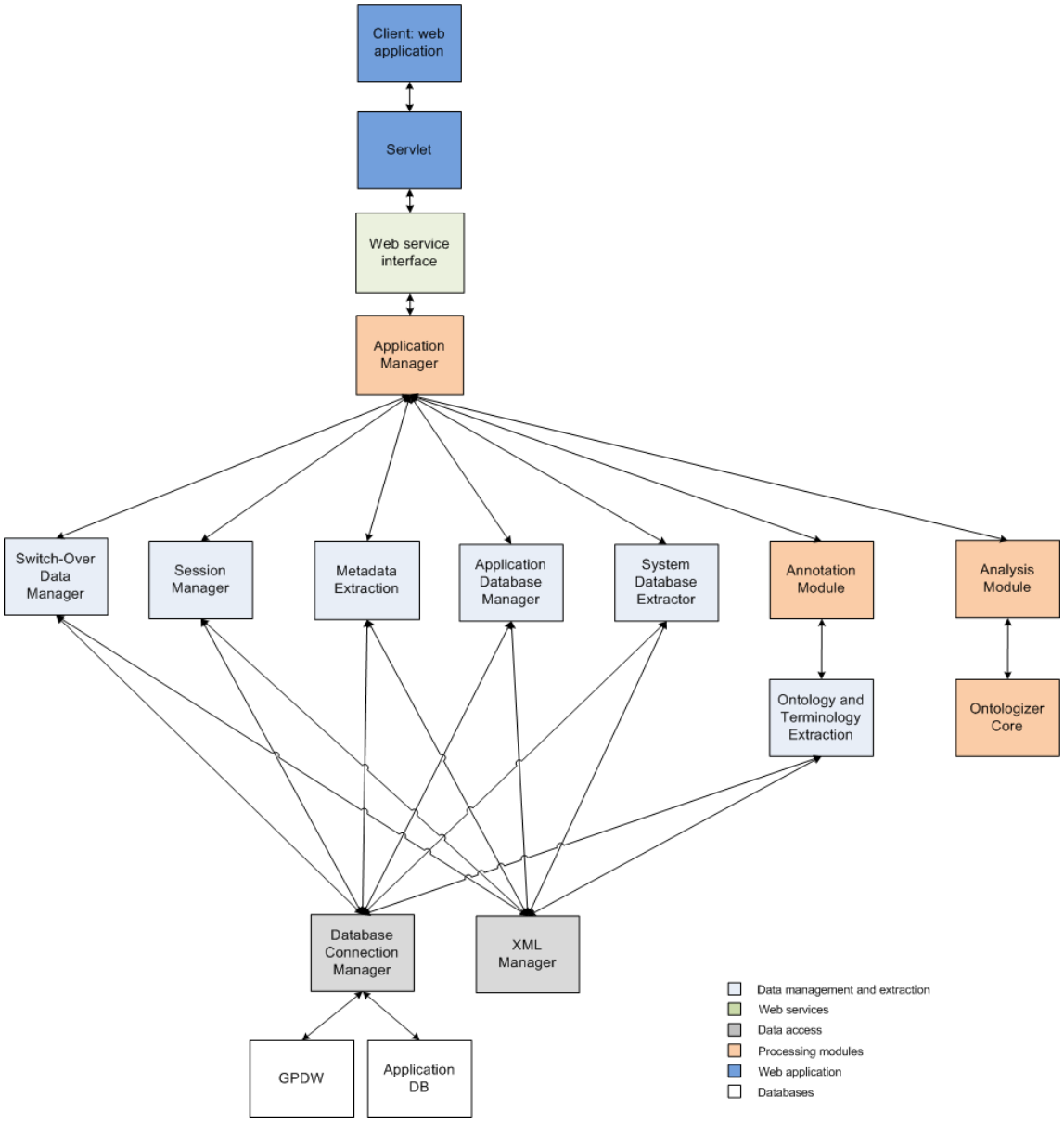


Figura 6.3. Schema dei componenti software del sistema

6.2 Gestione ed elaborazione dei dati

6.2.1 Uso dei dati nel Genomic and Proteomic Data Warehouse (GPDW)

Il Genomic and Proteomic Data Warehouse contiene un'enorme base di conoscenza di dati biomolecolari e biomedici. È stato realizzato utilizzando come DBMS PostgreSQL (attualmente si appoggia alla versione 8.4). All'interno del GPDW sono presenti centinaia di tabelle, molte delle quali contenenti svariati milioni di tuple. Inoltre la presenza di alcune tabelle dipende dall'avere o meno importato alcune sorgenti dati all'atto della compilazione del data warehouse. Risultano quindi evidenti i due maggiori problemi derivanti dalla realizzazione di un sistema informativo che utilizzi una base di conoscenza costruita in questo modo: la trattazione di grandi quantitativi di dati e la robustezza a variazioni dello schema di tale base di conoscenza. Il sistema è stato quindi progettato per essere in grado di gestire questi due aspetti.

6.2.1.1 Utilizzo dei metadati

Il primo problema considerato è quello della variazione dello schema dei dati. Il GPDW mette a disposizione alcune tabelle contenenti dei metadati (dati che descrivono altri dati) relativi alla base di conoscenza. Queste tabelle non variano all'aggiornamento del GPDW, quindi è possibile sfruttarle per ottenere informazioni sullo schema attuale della base di conoscenza in modo da poter poi recuperare i dati necessari al sistema per eseguire le interrogazioni.

Il sistema progettato, quindi, ogni volta che ha necessità di estrarre dei dati dalla base di conoscenza, effettua prima una query sulle tabelle dei metadati per ricavare le informazioni necessarie; in seguito esegue le query sulle tabelle dello schema principale del data warehouse. In questo modo, se ad esempio una sorgente dati non è stata importata all'interno del GPDW, la query sui metadati ne evidenzierà l'assenza. Questo procedimento evita di dover modificare il software a ogni aggiornamento del data warehouse. Di seguito è mostrato un esempio di query ai metadati che permette di ricavare le sorgenti dati disponibili per una certa feature (biomedica o biomolecolare):

```
SELECT source.source_name, source.source_id
FROM (metadata.feature JOIN metadata.source2feature ON
  feature.feature_id=source2feature.feature_id) JOIN metadata.source ON
```

```
  source.source_id=source2feature.source_id
WHERE feature.feature_name='gene'
```

La tabella *feature* contiene le informazioni su entità biomolecolari e feature biomediche; la tabella *source* contiene informazioni sulle sorgenti dati; la tabella *source2feature* contiene invece l'associazione fra le feature e le sorgenti dati.

6.2.1.2 Utilizzo dei numerosi dati biomolecolari

Il secondo problema trattato è quello di dover ridurre al minimo i dati utilizzati da ogni query e scambiati fra il sistema e il data warehouse. L'estrazione di migliaia di tuple è infatti un processo molto oneroso, soprattutto se fra i campi selezionati ve ne sono molti di tipo testo (meno se i campi sono memorizzati come numeri o stringhe di bit). È stato necessario quindi estrarre da ogni tabella il minor numero di dati possibili per ottenere un buon livello di performance del sistema. Sono stati estratti da ogni query, per quanto possibile, solo campi di tipo numerico o booleano, rinunciando alla maggior parte dei campi di tipo testo: per quanto riguarda la visualizzazione dei dati testuali da parte dell'utente, sono state effettuate delle query successive all'analisi, in modo da non dover mantenere tali dati in memoria durante l'elaborazione.

Un'altra ottimizzazione necessaria è stata la riduzione del numero di operazioni presenti in ogni query e il numero di tuple risultanti da esse: trattando dati così numerosi è infatti fondamentale limitare il più possibile il numero di JOIN fra tabelle, l'utilizzo di operatori aggregati, l'utilizzo di distinct.

Le parti in cui la gestione dei dati provenienti dalla base di conoscenza è più onerosa sono tre:

- la traduzione degli ID inseriti dall'utente negli OID del formato del data warehouse; infatti le tabelle di traduzione di geni, di proteine e di altre entità biomolecolari contengono tuple nell'ordine di decine di milioni, quindi portano a un rapido saturamento della memoria volatile allocata dal DBMS e a una riduzione delle performance
- l'estrazione di intere ontologie o terminologie; in particolar modo per quanto riguarda le ontologie, sono necessari diversi JOIN per ottenere le informazioni sulle relazioni fra i termini del grafo
- l'estrazione di annotazioni; anche questo procedimento richiede necessariamente alcuni JOIN fra tabelle molto grandi quali le tabelle di geni e proteine e quelle di annotazioni biomolecolari

Un altro importante aspetto da affrontare è l'overhead di comunicazione fra il sistema realizzato e il DBMS: se vengono effettuate molteplici query semplici il sistema rallenta in maniera molto superiore a quanto avverrebbe effettuando una sola query più complessa contenente però un maggior numero di dati. È per questo motivo che si è deciso di utilizzare le stored procedure, sfruttando il linguaggio integrato pl/pgSQL messo a disposizione da PostgreSQL. Infatti, soprattutto per quanto riguarda la traduzione di ID, l'applicazione deve poter convertire migliaia di ID per volta, ed effettuare migliaia di query in brevissimo tempo porta ad un evidente rallentamento. Chiamando invece una stored procedure, a cui vengono passate liste di migliaia di ID, nella comunicazione tra il sistema e il database vengono ridotti significativamente i costi di overhead; inoltre, grazie alle funzioni integrate nel DBMS, è possibile inserire queste liste in tabelle a costi molto ridotti; risulta poi efficiente l'utilizzo di queste tabelle generate a partire dalle liste per effettuare dei JOIN con altre tabelle presenti nel data warehouse. L'utilizzo delle stored procedure risulta molto efficace anche nel caso di estrazione di ontologie e di annotazioni: infatti, sebbene gli input iniziali non siano dati numerosi, utilizzando query semplici sarebbe necessario effettuarne diverse trasmettendo varie volte una grande quantità di dati fra sistema e database. Di seguito è mostrato un esempio di procedura realizzata in pl/pgSQL:

```
CREATE OR REPLACE FUNCTION generic_id_translation4(user_id character varying, list_id
        character varying, list text[], tableid character varying, entity character varying,
        id_trans_table character varying, source bit)
 RETURNS void AS
$BODY$
DECLARE
res int;
result record;
conv_table character varying:=quote_ident('biomolecular_id_'||user_id||'_'||list_id);
tmp_table character varying:=quote_ident('temp_biomolecular_id_'||user_id||'_'||list_id);
BEGIN
EXECUTE 'DROP TABLE IF EXISTS gpeaer.'||tmp_table;
EXECUTE 'DROP TABLE IF EXISTS gpeaer.'||conv_table;
EXECUTE 'DROP TABLE IF EXISTS '||quote_ident(tableID||'templist');
EXECUTE 'CREATE TEMP TABLE '||quote_ident(tableID||'templist')||'(id character varying)';
EXECUTE 'INSERT INTO '||quote_ident(tableID||'templist')||'(select * from
unnest('||quote_nullable(list)||'::text[]))';
EXECUTE 'CREATE TABLE gpeaer.'||tmp_table||' AS (
        SELECT '||quote_ident(id_trans_table)||'.translated_'||quote_ident(entity)||'_oid as
        entity_oid, '||quote_ident(tableID||'templist')||'.id as entity_id, source.source_xml_id,
         '||quote_ident(id_trans_table)||'.inferred,
        '||quote_ident(id_trans_table)||'.translated_'||quote_ident(entity)||'_concept_oid as
         concept_oid
        FROM '||quote_ident(tableID||'templist')||' LEFT JOIN '||quote_ident(id_trans_table)||' ON
```

```
        '||quote_ident(tableID||'templist')||'.id=
        '||quote_ident(id_trans_table)||'.'||quote_ident(entity)||'_id
        JOIN metadata.source ON
         source.source_id='||quote_ident(id_trans_table)||'.source_name
         WHERE '||quote_ident(id_trans_table)||'.source_name='||quote_nullable(source)||')';

EXECUTE 'CREATE TABLE gpeaer.'||conv_table||'(CHECK (username='||quote_nullable(user_id)||' AND
list_id='||quote_nullable(list_id)||')) INHERITS (gpeaer.biomolecular_id_'||quote_ident(user_id)||')';

EXECUTE 'INSERT INTO gpeaer.'||conv_table||'(username, list_id, entity_id, entity_source_xml_id,
inferred, concept_oid)
SELECT '||quote_nullable(user_id)||', '||quote_nullable(list_id)||', A.entity_id, A.source_xml_id,
A.inferred, A.concept_oid
        FROM gpeaer.'||tmp_table||' AS A';

RETURN;
END;$BODY$
 LANGUAGE 'plpgsql' VOLATILE;
```

La procedura mostrata effettua la traduzione di ID inseriti dall'utente in concept OID del formato del data warehouse e li memorizza in una tabella. I concept OID sono degli identificativi univoci generati dal data warehouse che rappresentano un concetto di entità biomolecolare, di caratteristica biomedica o di annotazione, indipendentemente dalla sorgente che fornisce tale dato. I risultati completi della traduzione non vengono restituiti al sistema, in modo da ridurre i tempi di esecuzione; quando sarà necessario accedere a tali dati sarà sufficiente passare al metodo che lo richiede l'identificativo della tabella che li contiene. Come si può notare, la procedura per la traduzione accetta diversi parametri di input, fra cui anche le liste di ID. Le query effettuate nella procedura devono essere dinamiche, infatti i nomi delle entità e delle tabelle coinvolte dipendono dalle scelte dell'utente: sono quindi dei parametri di input. Per questo motivo è necessario utilizzare il comando EXECUTE, che ha come argomento una stringa qualunque che verrà interpretata come statement SQL. La stringa da passare a EXECUTE viene quindi costruita sfruttando l'operatore di concatenazione di stringhe "||" fornito dal linguaggio. Per concatenare delle variabili nella stringa SQL è necessario utilizzare alcune funzioni fornite da PostgreSQL: la funzione *quote_ident()* fa in modo che la variabile venga considerata come un identificatore di tabella o di attributo; le funzioni *quote_literal()* e *quote_nullable()* servono invece a inserire delle vere e proprie variabili all'interno della stringa SQL.

6.2.1.3 Ottimizzazione delle query

Pur utilizzando delle stored procedures, non sempre l'estrazione dei dati risulta veloce: è stato quindi necessario effettuare dei passaggi di ottimizzazione anche per le singole query presenti nelle procedure. SQL offre il comando EXPLAIN, che permette di analizzare la sequenza di operazioni effettuate dal query planner: in questo modo è possibile tentare di ridurre, ad esempio, in numero di scansioni sequenziali su alcune tabelle, sfruttando ad esempio gli indici, oppure scambiare l'ordine di alcune operazioni all'interno della query in modo da ottenere un rendimento migliore.

```
CREATE TABLE templist(id character varying);
FOR index IN 1..list_size LOOP
        INSERT INTO templist values(list[index]);
END LOOP;
```

Ad esempio, nel frammento di codice precedente, viene eseguito un ciclo per inserire i valori di una lista passata in input in una tabella temporanea. Questo inserimento, però, non è efficiente: risulta molto più performante utilizzare una funzione integrata in PostgreSQL quale la *unnest*, come si può vedere nel frammento di codice seguente.

```
CREATE TABLE templist(id character varying);
INSERT INTO templist (select * from unnest(userlist));
```

I tempi di esecuzione variano infatti da 1,8 secondi per l'inserimento con ciclo for a 300 ms per l'inserimento tramite la *unnest*. È quindi conveniente usare ogni volta che sia possibile le funzioni messe a disposizione dal linguaggio pl/pgSQL.

Un'altra operazione che si è rivelata piuttosto dispendiosa è il *distinct*, oltre agli operatori aggregati: si è tentato di evitare tali operatori, ma il *distinct* è indispensabile per il sistema sviluppato. Perciò tale operatore è stato usato solo su tabelle ridotte al minimo, così da non inficiare le prestazioni.

Si è tentato sempre di ridurre il numero di tabelle coinvolte in operazioni di JOIN, e dove non è stato possibile, si è preferito spezzare le query troppo complesse sfruttando tabelle temporanee per memorizzare i risultati di query intermedie più semplici. Uno degli aspetti cruciali è stato inoltre la riduzione del numero di campi estratti per ogni query. La seguente query, ad esempio, risulta molto lenta.

```
SELECT distinct gene_id_translation.translated_gene_oid, id_table.id, gene.symbol
FROM id_table LEFT JOIN public.gene_id_translation ON templist.id=gene_id_translation.gene_id
LEFT JOIN public.gene ON gene_id_translation.translated_gene_oid=gene.gene_oid;
```

Per ottenere una sola informazione in più (*gene.symbol*), infatti, si è costretti a effettuare un JOIN supplementare con la tabella *gene*, che è molto grande. Per ottimizzare è

possibile invece estrarre solo i dati veramente indispensabili (gli OID) e recuperare altri attributi in seguito, al momento della visualizzazione dei risultati da parte degli utenti, dato che tali attributi non servono durante l'elaborazione.

6.2.2 Gestione dei dati dell'utente

Per la gestione dei dati del sistema è stato necessario l'utilizzo di uno schema all'interno della base di conoscenza: infatti molti dei dati memorizzati devono poter interagire con i dati presenti nel GPDW, e PostgreSQL non permette di effettuare query inter-database. Per la memorizzazione dei dati che non necessitano di interagire con la base di conoscenza è stato implementato un database separato. I diagrammi E-R dei database progettati saranno illustrati nei paragrafi seguenti; gli schemi logici di tali database e le versioni ristrutturate di alcuni diagrammi E-R sono mostrati in Appendice.

6.2.2.1 Database del sistema sviluppato

Nel database proprio dell'applicazione (in figura 6.4), sono presenti una tabella contenente i dati personali degli utenti, USER, una tabella per i log, una tabella contenente i dati delle sessioni e alcune tabelle per memorizzare dati necessari a una procedura che gestisce l'aggiornamento degli schemi di dati del sistema presenti nel GPDW (illustrata nel capitolo 6.4 "*Gestione dell'aggiornamento della base di conoscenza"*). L'attributo *Type* dell'entità USER assume valori diversi in base alla tipologia di utente (standard, avanzato, amministratore). Sempre nella stessa entità, l'attributo *Is Enabled* indica se l'utente è abilitato o meno all'accesso al sistema. L'entità LAST SESSION rappresenta l'ultima sessione effettuata da un utente nel sistema: contiene le informazioni sugli istanti di inizio e fine sessione, di ultima azione e di ultima modifica ai propri dati. Il significato delle entità TABLE TO UPDATE, OPERATION e FILTER ATTRIBUTE e degli attributi *Last Connection* e *Working during switch over* dell'entità USER verranno illustrati nel capitolo 6.4, dedicato alla gestione dell'aggiornamento della base di conoscenza.

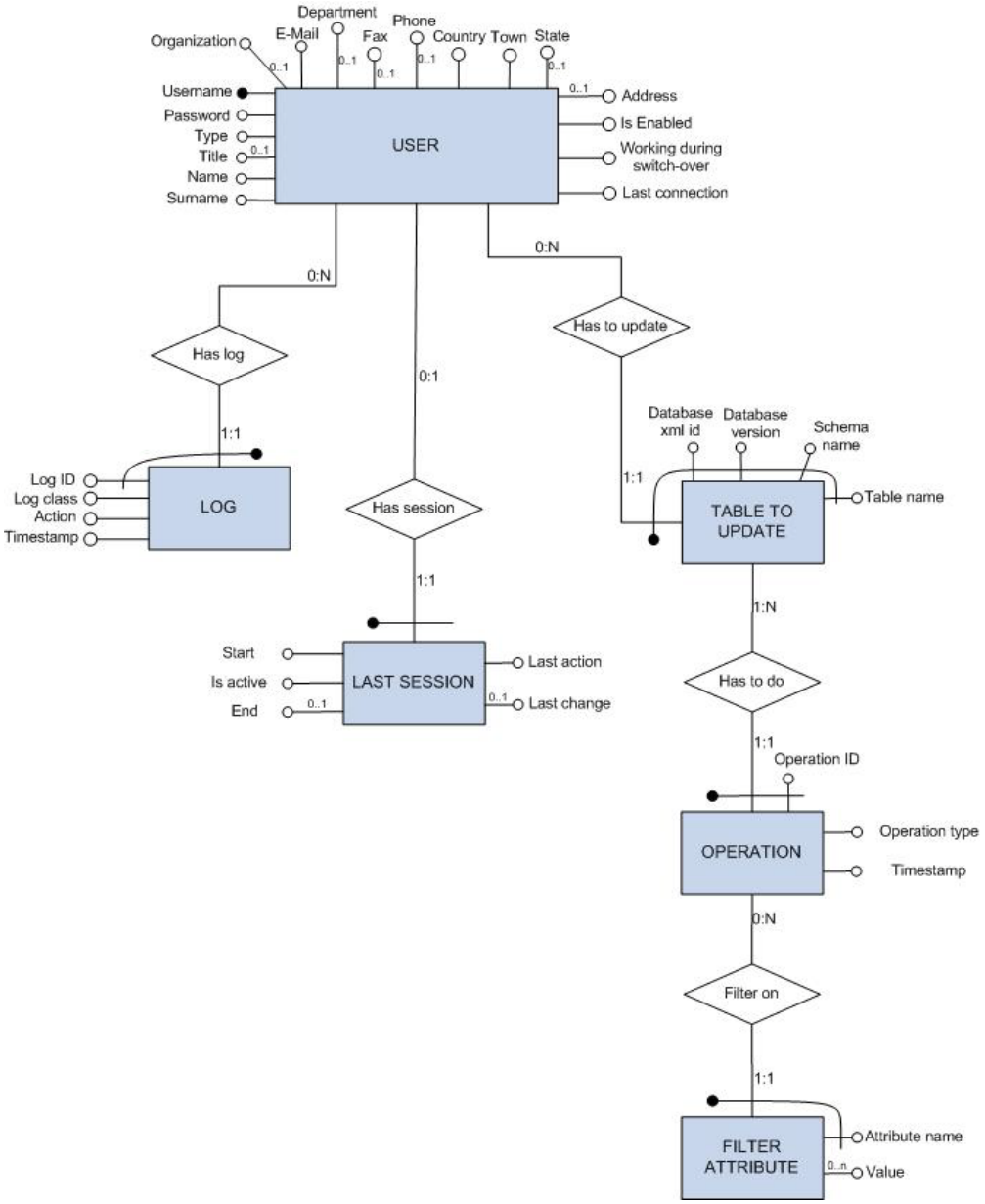


Figura 6.4. Schema E-R del database del sistema

6.2.2.2 Schema dei dati del sistema nel GPDW

Tenendo conto delle esigenze esposte in fase di definizione dei requisiti, è stato necessario aggiungere al database di back-end uno schema di dati da inserire all'interno del GPDW, poiché alcuni dati memorizzati dagli utenti dipendono o interagiscono con quelli della base di conoscenza. Lo schema E-R è illustrato in figura 6.5. È presente un'entità USER, che contiene solamente gli username degli utenti, mentre i dati personali sono memorizzati nel database separato, come detto in precedenza. La presenza di questa tabella è necessaria per consentire l'applicazione di vincoli di integrità a tutte le tabelle dello schema. È possibile memorizzare delle liste di identificativi (per utenti avanzati): le informazioni generali sulla lista vengono salvate in LIST, tutti gli ID appartenenti a essa vengono memorizzati in BIOMOLECULAR ID, insieme ai loro dati specifici e al concept OID del formato del data warehouse corrispondente. Si è scelto di memorizzare solo i concept OID e non anche gli OID per ridurre lo spazio occupato nel database. Se fossero stati memorizzati solo gli OID, sarebbe stata necessaria una conversione in concept OID per le analisi di tipo concettuale (che sono quelle predefinite nel sistema) che richiede un tempo piuttosto lungo; si è deciso quindi di memorizzare i concept OID, consci del fatto che finché non sarà disponibile il livello di integrazione concettuale nel GPDW sarà necessario riconvertirli in OID per l'annotazione, ma una volta che tale livello di dati sarà disponibile questa struttura permetterà di gestire le analisi efficientemente. L'utente può inoltre memorizzare diversi valori di classificazione per ogni ID caricato: questi valori vengono salvati in CLASSIFICATION. L'utente può inoltre salvare degli esperimenti (in EXPERIMENT), ognuno dei quali può contenere uno o più sottocasi. Le relazioni *In* P*opulation set* e *In Study set* servono ad associare le liste selezionate per un certo caso dell'esperimento e i valori di classificazione scelti. Nello schema del database sarebbe possibile memorizzare più liste facenti parte di un solo population set (o di un solo study set); nel sistema realizzato però questa caratteristica non è stata ancora implementata, ma ciò avverrà in futuro, per cui allo stato attuale può essere scelta solo una lista per population set e una per study set (anche con molteplici valori di classificazione). Per ogni caso dell'esperimento possono essere effettuate più analisi (le cui impostazioni generali sono memorizzate in ANALYSIS); durante un'analisi di arricchimento possono essere analizzate più caratteristiche biomediche (le impostazioni relative all'analisi per ognuna di esse sono memorizzate in FEATURE ANALYSIS). Le sorgenti dati considerate nell'analisi per ogni feature biomedica sono salvate in

SOURCE ANALYZED; i risultati dell'analisi sono memorizzati in RELEVANT TERM; se è stata effettuata un'analisi di tipo concettuale lato feature biomedica saranno presenti in ASSOCIATED TERM gli ID dei termini di feature biomedica corrispondenti allo stesso concetto di un termine di feature salvato in RELEVANT TERM. L'attributo *Significance value* rappresenta il p-value corretto con metodi per la correzione di test multipli oppure, nel caso sia stata effettuata un'analisi di tipo MGSA, rappresenta il valore della probabilità marginale calcolata. Poiché i dati dipendenti dalla versione specifica della base di conoscenza dovranno essere aggiornati a ogni cambiamento di versione, si è tentato di minimizzare il loro numero all'interno dello schema dei dati del sistema. Sono stati memorizzati, per quanto possibile, dei dati stabili quali gli xml_id relativi a feature e sorgenti (ricavabili dalle tabelle dei metadati che, essendo storiche, non risentono dell'aggiornamento) e gli ID di entità e feature nel formato delle sorgenti importate anziché gli OID nel formato del GPDW. L'unica eccezione è stata fatta per la memorizzazione dei concept OID in BIOMOLECULAR ID: ciò è stato necessario per ridurre i tempi di esecuzione evitando di effettuare la fase di traduzione a ogni riutilizzo della lista. I dati relativi ai concept OID dovranno quindi essere aggiornati a ogni cambiamento di versione della base di conoscenza. Per la progettazione del database è stato considerato che ogni utente possa utilizzare solo ed esclusivamente i propri dati (liste di ID, impostazioni, risultati delle analisi).

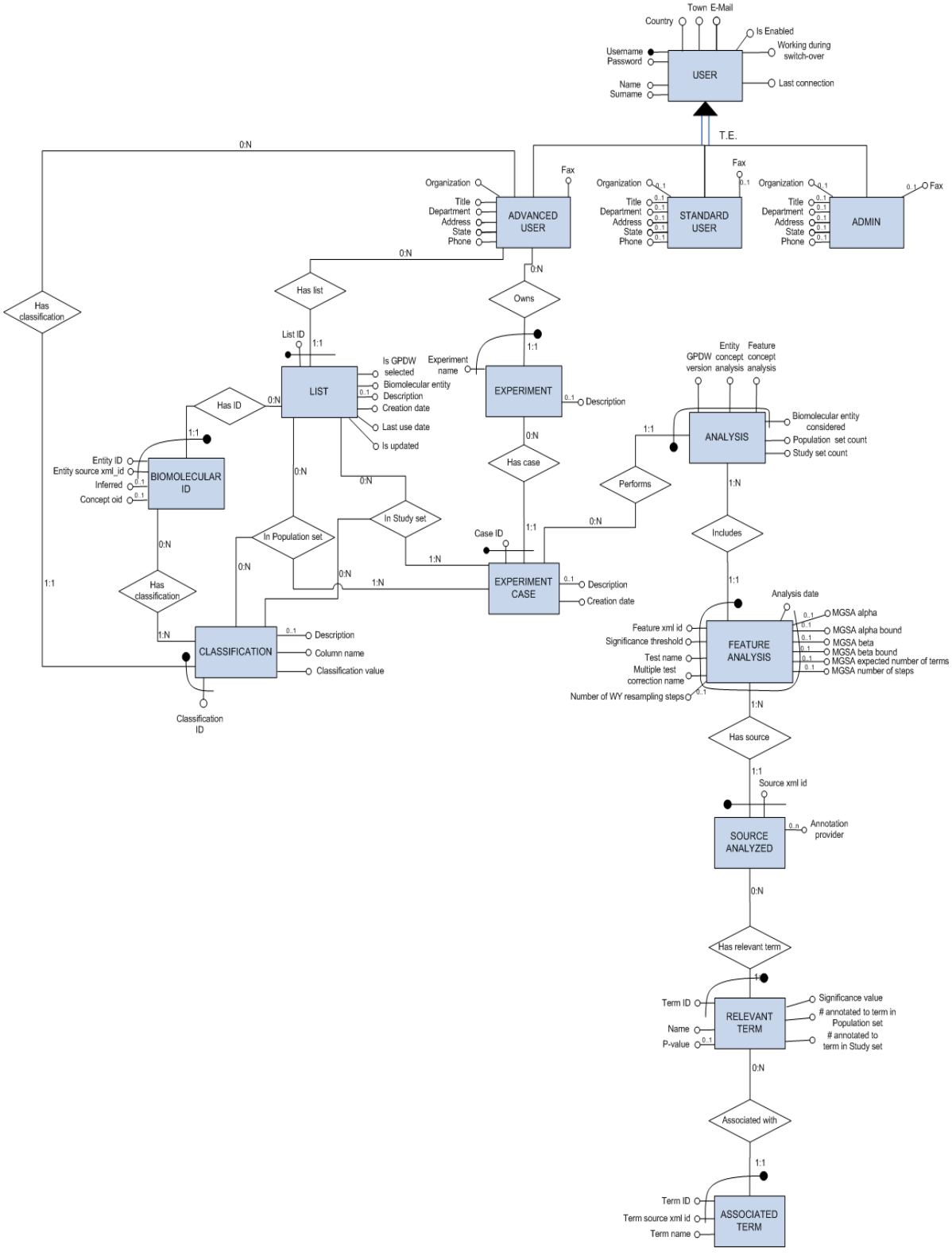


Figura 6.5. Diagramma E-R dello schema dei dati del sistema nel GPDW

Le tabelle contenenti gli ID caricati dall'utente e i risultati delle analisi tendono subito ad assumere dimensioni elevate: tipicamente infatti gli utenti caricheranno liste da migliaia di ID, e anche i risultati possono essere migliaia di tuple per ogni analisi. Un

aumento così rapido delle dimensioni delle tabelle comporta senza dubbio un rallentamento nell'interrogazione e nell'estrazione di dati: per evitare questa situazione è stato sfruttato il meccanismo di partizionamento offerto da PostgreSQL. Le tabelle contenenti ID e risultati sono state strutturate in questo modo: una tabella logica che consente l'accesso a tutti i dati e una tabella fisica per ogni lista di ID o di risultati; in ogni tabella fisica deve essere dichiarato un vincolo mutuamente esclusivo rispetto agli altri. Poiché anche l'eccessivo numero di partizioni per una sola tabella porta degli svantaggi, è stato implementato un partizionamento a due livelli: al primo livello le tabelle sono state partizionate per utente, al secondo livello per singola lista di ID o risultati del dato utente. Per la ricerca di dati su una tabella partizionata il DBMS deve eseguire la valutazione di tutti i vincoli presenti nelle partizioni, anche se già ne è stato trovato uno soddisfatto, e questo, superato un certo numero di partizioni, può portare più svantaggi che vantaggi [44]; utilizzando il meccanismo illustrato, verranno dapprima valutati i vincoli sugli utenti, poi quelli sulla singola lista: si passa quindi dal dover valutare N vincoli (nel caso di partizionamento a un livello) a N/u vincoli (dove con N è indicato il numero totale di partizioni e con u il numero degli utenti che hanno dei dati memorizzati nella tabella).

Per quanto riguarda i dati temporanei vengono utilizzate delle tabelle temporanee SQL, che vengono automaticamente eliminate al termine della connessione al database. Se questo può andare bene per tabelle che servono a una singola operazione, non è così per tabelle che devono continuare ad esistere per tutta la durata della sessione dell'utente nel sistema, poiché, essendo il sistema basato su un architettura orientata ai servizi di tipo REST, non esistono vere e proprie sessioni a livello applicativo, ed ogni chiamata ai servizi è stateless. Questo problema si verifica soprattutto per quanto riguarda gli utenti standard: infatti essi non possono salvare dati nella struttura fissa del database (si è scelta questa opzione per evitare un'esplosione della dimensione del database, dato che alcune tabelle conterranno diverse centinaia di migliaia di tuple), ma devono comunque avere a disposizione delle tabelle con le liste caricate e tradotte durante la sessione. Si è scelto quindi di salvare questo tipo di dati in tabelle non temporanee che vengono cancellate all'atto della chiusura della sessione dell'utente. Si è posto poi il problema delle sessioni che scadono senza che l'utente abbia effettuato il logout: per gestire questo caso è stata implementata una procedura di rimozione delle tabelle inutilizzate (che rimane attiva in background) che verrà illustrata nel capitolo 6.3.7.1.

6.2.3 Gestione di liste di ID di entità biomolecolari

Il sistema è stato progettato per supportare l'inserimento di una o due liste per volta. Si può caricare una sola lista per effettuare un'analisi a patto che contenga ID con almeno due valori di classificazione diversi (necessari per distinguere lo study set dal population set). Altrimenti è possibile caricare una lista che rappresenti il population set e una lista che rappresenti lo study set. Al caricamento della lista sarà necessario specificare l'entità biomolecolare e la sorgente dati di appartenenza degli ID. La fase di gestione delle liste di ID si divide in due operazioni principali. La prima è il controllo di conformità delle liste caricate: vengono segnalati all'utente ed eliminati errori grossolani quali la presenza di ID duplicati in una lista e viene segnalato all'utente se vi sono sovrapposizioni fra le liste caricate (se ne vengono caricate più di una). Viene inoltre effettuato un controllo sul formato degli ID inseriti: grazie a una tabella dei metadati del GPDW contenente le espressioni regolari associate al formato degli ID di tutte le principali banche dati biomolecolari è stato implementato un controllo che permette il riconoscimento della sorgente di appartenenza di ogni ID; vengono segnalati all'utente ID non conformi alla sorgente di provenienza dichiarata per la lista caricata. È possibile che gli ID caricati siano però solo leggermente difformi dalla sintassi corretta: in casi come segni di punteggiatura in eccesso o inserimento di caratteri maiuscoli anziché minuscoli, la procedura di controllo sviluppato permette di correggere tali errori.

La seconda fase di gestione delle liste consiste nella traduzione da ID inserito dall'utente (nel formato di una certa sorgente dati) a concept OID nel formato della base di conoscenza. L'operazione di traduzione avviene come segue: gli ID caricati (già depurati di eventuali errori) sono passati in input ad una stored procedure che li inserisce in una tabella temporanea; viene quindi effettuata un'operazione di JOIN fra tale tabella e la tabella integrata di *id_translation* dell'entità biomolecolare selezionata dall'utente per estrarre i concept OID. Per gli utenti avanzati i risultati di traduzione vengono salvati nella tabella fissa del database contenente le liste di ID. Eventuali ID di cui non è stato trovato un corrispondente OID vengono segnalati all'utente: tali ID possono essere o riconosciuti come obsoleti o non riconosciuti nella base di conoscenza (quindi potrebbero essere ID di una sorgente dati non importata oppure ID più recenti dell'ultimo aggiornamento della base di conoscenza). Gli ID senza un corrispondente concept OID sono comunque memorizzati dal sistema.

6.2.4 Gestione di annotazioni biomolecolari

È stato implementato un componente software per l'estrazione di annotazioni a partire dalle liste di ID di entità biomolecolari caricate dall'utente. Per effettuare l'analisi statistica (che verrà trattata nel capitolo 6.2.5) sono necessari un population set, uno study set, l'elenco di tutte le annotazioni biomolecolari relative a essi e l'intera terminologia (o ontologia) che si è scelto di analizzare. Per prima cosa è stato quindi necessario sviluppare una classe Java per l'estrazione di intere terminologie o ontologie. Per quanto riguarda le terminologie il procedimento è molto semplice, si tratta infatti di una semplice query sulla tabella integrata presente nella base di conoscenza che restituisce i dati direttamente alla classe Java. L'estrazione di intere ontologie è invece più complessa: infatti è necessario estrarre, oltre ai dati relativi ai singoli termini di feature biomedica, anche le relazioni fra essi. Quindi è stata realizzata una stored procedure che recuperasse i dati dei termini e delle relazioni dalla tabella di feature dell'ontologia e dalla tabella di *relationship* nel GPDW, che contiene le associazioni fra un termine e tutti i suoi figli nel grafo. Per realizzare analisi di tipo concettuale lato feature biomedica, è stato necessario un ulteriore passaggio: poiché l'analisi concettuale analizza tutti i termini di una certa feature indipendentemente dalla sorgente dati di appartenenza, all'atto dell'estrazione vengono recuperate N ontologie (una per ogni sorgente). Per realizzare un'analisi sensata, è stato realizzato un metodo che fondesse tutte le ontologie di varie sorgenti in una unica preliminarmente all'estrazione dei dati. La stored procedure sviluppata per le ontologie salva le informazioni su esse in una tabella temporanea. Una volta estratte le informazioni su ontologie o terminologie, vengono inserite in alcune strutture dati messe a disposizione da Ontologizer: vi sono alcune strutture utili a memorizzare i termini dei vocabolari controllati e una struttura che serve a rappresentare un grafo, e che mette a disposizione dei metodi per attraversarlo. Una volta terminato questo passo, il sistema richiama una stored procedure che estrae e salva in una tabella temporanea tutte le annotazioni per le liste di ID utilizzate nell'analisi filtrando i risultati secondo alcuni criteri eventualmente selezionati dall'utente (sorgente specifica di una feature biomedica o provider delle annotazioni biomolecolari). La classe che si occupa della gestione delle annotazioni, quindi, estrae i dati da tale tabella e inserisce le informazioni relative alle annotazioni in strutture dati apposite (strutture messe a disposizione sempre da Ontologizer). Finora si

è parlato sempre di estrarre annotazioni relative a una singola feature biomedica: l'annotazione e l'analisi vengono effettuate per singola feature, ma è possibile elaborare più feature simultaneamente istanziando un thread per ognuna.

6.2.5 Analisi statistiche: Ontologizer

Quando tutti i dati necessari sono disponibili, il componente per la gestione delle annotazioni passa il controllo al componente per l'analisi. Questo componente richiama le funzioni già implementate in Ontologizer. Vengono passati come parametri di input population set e study set, il grafo rappresentante l'ontologia o la terminologia, la struttura contenente le annotazioni, il test statistico e la correzione per test multipli scelti. Le classi di Ontologizer effettuano quindi l'analisi e restituiscono una lista di termini del vocabolario controllato selezionato con associati i valori di significatività calcolati. Nel caso degli algoritmi basati su Test di Fisher i risultati sono i p-value e i p-value corretti calcolati; nel caso l'analisi selezionata sia di tipo MGSA (quindi basata su reti bayesiane) i risultati restituiti sono delle marginali, quindi da interpretare in maniera opposta ai valori di p-value: un termine è tanto più significativo quanto la marginale ha un valore elevato (mentre p-value più bassi rappresentano significatività più elevate).
È stato deciso di utilizzare Ontologizer per le analisi statistiche poiché offre un discreto numero di algoritmi ben ottimizzati (fra cui alcuni con correzioni specifiche per dipendenze fra termini di ontologie) e quindi risulta essere veloce nelle analisi; sono state però necessarie alcune modifiche per adattarlo alle esigenze di questa Tesi.

6.2.5.1 Estensione di Ontologizer

La prima modifica da implementare è stato il supporto ad un tipo di ID diverso da quello della Gene Ontology (gli unici ufficialmente riconosciuti da Ontologizer): per il sistema sviluppato in questa Tesi vengono infatti utilizzati nell'analisi gli OID nel formato del GPDW. Sono state quindi modificate le classi TermID e Association, che rappresentano rispettivamente l'ID di un termine di una feature biomedica e l'annotazione fra entità biomolecolari e feature biomediche, introducendo nuovi costruttori.
La seconda modifica è stata l'implementazione di una variazione del Test di Fisher: come già accennato nel capitolo 5.5.2, Ontologizer sfrutta il Test di Fisher semplice in

versione a una coda per effettuare l'analisi di arricchimento. Sarebbe invece più indicato, per l'esecuzione di questo tipo di analisi, l'utilizzo del Test di Fisher a due code con mid-p-value [45]. Il calcolo con mid-p-value consiste nel dimezzare il valore della probabilità di osservare valori uguali a quelli attesi sotto l'ipotesi nulla. Si è scelto quindi di implementare questa versione del test utilizzando il metodo della duplicazione del valore del test a una coda per ottenere il valore del test a due code: ciò non inficia la qualità del risultato ma velocizza l'esecuzione e semplifica l'implementazione [7].

6.2.5.2 Protocollo di aggiornamento di Ontologizer

Poiché Ontologizer è un software in costante sviluppo, è vantaggioso tenere aggiornare le classi utilizzate nel sistema sviluppato in questa Tesi all'ultima versione disponibile, in modo da godere di vantaggi derivanti da miglioramenti realizzati negli algoritmi. È stata schematizzata una procedura per effettuare questo aggiornamento:

1. verificata la presenza di aggiornamenti, scaricare i file dall'SVN di Ontologizer (all'indirizzo web https://ontologizer.svn.sourceforge.net/svnroot/ontologizer) utilizzando un programma apposito di subversion e ponendo tali file in una directory a scelta (è sufficiente scaricare la cartella "trunk"). Ciò va fatto con privilegi di amministratore, se si utilizza Windows Vista o 7.
2. copiare le sottocartelle
    - trunk\ontologizer\src.grappa\att
    - trunk\ontologizer\src.grappa\java_cup
    - trunk\ontologizer\src\env
    - trunk\ontologizer\src\ontologizer
    - trunk\ontologizer\src\sonumina
    - trunk\ontologizer\src\tools

    nella cartella GPEAer\src contenente i sorgenti del sistema sviluppato.
3. rinominare le classi Hypergeometric.java presente nella sotto-cartella "\src\ontologizer" , Association.java presente nella sottocartella "\src\ontologizer\association", TermID.java presente nella sottocartella "\src\ontologizer\go" della cartella "GPEAer\src" in Hypergeometric2.java, Association2.java, TermID2.java
4. aggiungere nelle stesse cartelle le classi Hypergeometric.java, Association.java, TermID.java sviluppate in questa Tesi

5. inserire le classi GetOntology.java e GetTerminology.java sviluppate in questa Tesi nella sottocartella "\src\ontologizer\go"
6. inserire in "GPEAer\src" le cartelle contenenti i file sorgenti del sistema sviluppato
7. compilare tutte le classi presenti nella cartella "GPEAer\src"
8. effettuare il deployment su web server.

6.3 Servizi ed applicazione web sviluppati

Il sistema sviluppato in questa Tesi deve permettere l'utilizzo di tutte le funzionalità implementate al maggior numero di utenti possibili e l'integrazione con altri applicativi. Sono stati quindi realizzati dei servizi web per l'esposizione di queste funzionalità e un'applicazione poggiata su essi. Nella realizzazione dei servizi è stato seguito il paradigma REST, per cui essi sono raggiungibili utilizzando i semplici metodi definiti dallo standard HTTP. L'input e l'output di tali servizi possono essere sia dei frammenti di codice XML sia dei frammenti di codice JSON, in base alla scelta effettuata dal chiamante. Ovviamente, per essere utilizzati all'interno di altri software, sono necessarie operazioni di serializzazione e de serializzazione degli oggetti Java in XML o JSON: ciò viene effettuato in automatico nel sistema realizzato grazie all'implementazione di strutture dati integrate con le annotazioni JAXB.
I servizi realizzati potranno essere utilizzati da altri sviluppatori che vogliono costruire delle applicazioni web sulla loro base; potranno essere inclusi anche in workflow di servizi bioinformatici, ad esempio utilizzando programmi come Taverna [46]; potranno inoltre essere inclusi in sistemi di search computing come SeCO [47] (un progetto di ricerca sul search computing portato avanti dal Politecnico di Milano).
In particolare i servizi sviluppati si dividono in tre categorie: servizi di look-up, servizi per la gestione del database e i servizi di elaborazione.

6.3.1 Servizi di look-up

Mediante questi servizi è possibile recuperare informazioni necessarie in seguito come input per i servizi di elaborazione. Sono servizi molto semplici, che accettano in input dei parametri di tipo stringa e in output restituiscono delle stringhe o delle liste di

stringhe. Un esempio di questi servizi è il recupero di tutti i nomi delle sorgenti dati associate a una data entità biomolecolare o feature biomedica:

```
@GET
@Path("sources")
@Produces( { "application/xml", "application/json" })
@Consumes("text/plain")
public MyList availableSources(@QueryParam("feature")
       String feature) {…}
```

In questo frammento di codice vengono mostrate tutte le principali annotazioni JAXB utilizzate. L'annotazione @GET indica che il metodo può essere richiamato tramite una GET HTTP; l'annotazione @Path denota il percorso relativo a cui è raggiungibile il servizio: se ad esempio il percorso base è http://localhost:8080/GPEAer/services/, questo servizio sarà disponibile all'indirizzo http://localhost:8080/GPEAer/services/sources/. Le annotazioni @Produces e @Consumes indicano i formati di input accettati e i formati di output disponibili: in questo caso in input viene accettato un tipo testo, mentre in output è possibile ottenere del codice XML o JSON. L'annotazione @QueryParam associa i parametri della funzione Java a quelli della richiesta HTTP specificandone il nome. Come si può notare il tipo di ritorno del metodo non è un tipo predefinito di Java: è infatti una classe che è stata creata per contenere una lista di stringhe (poiché le liste presenti nelle librerie di Java non sono fra i tipi supportati nativamente da JAXB). Di seguito viene mostrato un frammento della classe MyList:

```
@XmlRootElement(name="Values")
public class MyList{
      @XmlElement(required=true)
      public List<MyString> data;

      public MyList() {}
```

L'annotazione @XmlRootElement associa la classe Java al nodo radice di un albero XML. È poi possibile annotare i vari campi della classe Java con @XmlElement o @XmlAttribute, in modo che vengono rispettivamente associati a un elemento o un attributo di un elemento XML. Si può notare che la lista memorizzata in questa classe non è di stringhe ma di un tipo MyString: infatti, è necessario annotare tutti i tipi che si desidera serializzare; poiché il tipo String di per sé non è serializzabile, è stato creato il wrapper MyString che ne permette il marshalling. Tutti gli altri servizi di look-up implementati seguono questo schema.

6.3.2 Servizi per la gestione del database del sistema

Sono stati implementati dei servizi anche per la memorizzazione di dati nel database e per la loro estrazione. I servizi per la memorizzazione, non restituendo alcun valore, sono richiamabili tramite la primitiva PUT HTTP. Per l'eliminazione di dati dal database va utilizzata invece la primitiva DELETE. Più interessanti sono invece i servizi per l'estrazione di dati, che sono stati implementati avvalendosi di un'ulteriore caratteristica di JAX-RS: le annotazioni di percorso.

```
@GET
@Path("/list/{username}/{listName}")
@Produces( { "application/xml", "application/json" })
@Consumes( { "application/xml", "application/json" })
public MyIDList getListInfo(@PathParam("username") String
        username,@PathParam("listName") String listName) {}
```

Come si può vedere, il contenuto dell'annotazione @Path è diverso dai frammenti di codice illustrati nel paragrafo precedente; viene infatti specificato un percorso dinamico. I termini fra parentesi graffe sono parametri passati in input, e corrispondono ai parametri del metodo Java (annotati infatti con @PathParam). Ciò significa che è possibile aggiungere all'URL i parametri necessari come se fossero directory da attraversare. In questo caso, specificando il nome dell'utente e il nome di una lista (ad esempio /list/marco/lista1), il servizio restituisce l'elenco degli ID appartenenti a essa con le classificazioni associate.

6.3.3 Servizi di elaborazione dati

I servizi di elaborazione offerti sono principalmente tre: uno per il controllo di conformità delle liste, uno per la traduzione degli ID e uno per l'analisi di arricchimento. Questi servizi non si differenziano molto dai precedenti; le uniche varianti consistono nel mapping con il metodo POST HTTP (non PUT poiché devono restituire dei risultati) e nel passaggio di parametri di tipo Entity nella richiesta. Ogni metodo ammette al massimo un parametro di tipo Entity per richiesta (in combinazione con un numero arbitrario di parametri di altro tipo): tale parametro rappresenta il message body della richiesta HTTP e non necessita di annotazioni JAXB. Nel frammento di codice sottostante è mostrato il prototipo della funzione per l'esecuzione dell'analisi di arricchimento.

```
@POST
@Produces( { "application/xml", "application/json" })
@Consumes( { "application/xml", "application/json" })
public AllResults executeAnalysis(@QueryParam("username") String
username, @QueryParam("password") String password,
                  ApplicationParameterBean appParam){}
```

6.3.4 Chiamata ai servizi web

Per richiamare i servizi web sono state utilizzate delle librerie di Apache CXF che implementano le specifiche presenti in JAX-RS relativi ai client HTTP-based. Ecco un esempio di codice che richiama il servizio di analisi:

```
WebClient client=WebClient.create(url);
client.query("username", username);
client.query("password", password);
client.accept("application/xml");
AllResults allres = client.post(appParam,AllResults.class);
```

Viene creato un oggetto di tipo WebClient (che rappresenta un Client HTTP) tramite un metodo statico che accetta in input un indirizzo URL. Il metodo *query* permette di aggiungere parametri alla richiesta HTTP tramite un mapping fra nome del parametro e variabile passata. Il metodo *accept* indica al servizio il formato in cui si vuole ottenere la risposta. Si passa quindi all'invocazione vera e propria del servizio con il metodo *post*: in input sono passati un parametro complesso che rappresenta il message body HTTP (in questo caso *appParam*) e il tipo di dato atteso come risultato (la classe *AllResults* implementata rappresenta una serie di liste contenenti i risultati di analisi su varie feature biomediche). L'invocazione degli altri servizi avviene in maniera del tutto analoga, salvo qualche variazione dovuta al diverso metodo HTTP utilizzato e alla presenza o meno di parametri nella richiesta.

6.3.5 Applicazione web

In questo lavoro di Tesi è stata anche sviluppata un'applicazione web poggiata sui servizi realizzati. L'applicazione offre un'interfaccia user-friendly adatta anche a utenti senza particolari conoscenze informatiche e sfrutta tutte le funzionalità messe a disposizione dai servizi: caricamento, traduzione e salvataggio di liste di identificativi biomolecolari, definizione di esperimenti, analisi di arricchimento su molteplici vocabolari controllati, salvataggio e riutilizzo di impostazioni di analisi e risultati. L'applicazione web è stata realizzata servendosi delle tecnologie JSP e Java Servlet. Le

servlet sviluppate hanno lo scopo di gestire le richieste degli utenti richiamando i servizi web appropriati. All'interno delle servlet non vengono effettuate particolari elaborazioni, salvo la necessaria conversione di formato di alcuni parametri per conformarli ai requisiti dei servizi web.

6.3.5.1 Client web

L'applicazione sviluppata non ha particolari requisiti per il client: per utilizzarla è sufficiente disporre di un normale browser per la navigazione web che deve essere abilitato all'esecuzione di codice Javascript. Javascript è utilizzato infatti per effettuare dei controlli preliminari sui dati inseriti dall'utente (all'interno delle form HTML), e soprattutto per effettuare chiamate asincrone alla servlet (tramite AJAX) per ottenere dati da visualizzare dinamicamente all'interno della pagina web. Le chiamate AJAX vengono inoltrate alla servlet, che lancia un'invocazione a un servizio web e restituisce risultati alla pagina web. Questo meccanismo viene sfruttato per poter presentare agli utenti le scelte disponibili per alcune impostazioni ricavando tali dati dalla base di conoscenza.

6.3.6 Configurazione del sistema

Per rendere il sistema configurabile senza la necessità di modificare il codice sorgente sono state esportate tutte le impostazioni in file XML. Sono presenti quattro file di configurazione.

Il file config.xml contiene le informazioni sui database utilizzati dal sistema: indirizzo, credenziali di accesso, schemi presenti. Nel file queries.xml sono esportate tutte le query utilizzate all'interno del sistema (le variabili nelle query dinamiche sono sostituite da un simbolo placeholder). Il file configparams.xml contiene i parametri di default per l'analisi statistica. Infine, il file update.xml contiene l'elenco degli schemi di database da ripristinare durante l'aggiornamento della base di conoscenza (problema che verrà trattato nel capitolo 6.6). Di seguito è mostrato un frammento del file XML contenente i parametri di default dell'analisi di arricchimento:

```
<?xml version="1.0" encoding="UTF-8"?>
<params xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="configparams_xsd.xsd">
      <WY>
            <numresampling value="500" />
      </WY>
```

```
      <MGSA>
            <alpha value="10" />
            <beta value="25" />
            <alphabound value="100" />
            <betabound value="100" />
            <expectedterms value="5" />
            <numsteps value="500000" />
      </MGSA>
</params>
```

Tutti i documenti XML creati sono stato validati con uno schema XSD.

Un altro passaggio necessario è stato la configurazione del sistema per l'esposizione dei servizi web su server. Sono state utilizzate le librerie del framework Spring (incluse in Apache CXF) per l'esposizione dei servizi; nei frammenti di codice sottostante sono mostrati i passaggi significativi dei file di configurazione XML:

-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
 <display-name>GPEAer</display-name>

 <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>WEB-INF/beans.xml</param-value>
</context-param>

<listener>
      <listener-class>
      org.springframework.web.context.ContextLoaderListener
      </listener-class>
</listener>

<servlet>
  <display-name>CXFServlet</display-name>
  <servlet-name>CXFServlet</servlet-name>
  <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-
class>
  <load-on-startup>1</load-on-startup>
 </servlet>

<servlet-mapping>
  <servlet-name>CXFServlet</servlet-name>
  <url-pattern>/services/*</url-pattern>
 </servlet-mapping>

</web-app>
```

-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  xsi:schemaLocation="
http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxrs
http://cxf.apache.org/schemas/jaxrs.xsd">

  <import resource="classpath:META-INF/cxf/cxf.xml" />
  <import resource="classpath:META-INF/cxf/cxf-extension-jaxrs-
binding.xml" />
  <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />


  <jaxrs:server id="enrichmentAnalysisService" address="/">
    <jaxrs:serviceBeans>
      <ref bean="enrichmentAnalysisResource" />
    </jaxrs:serviceBeans>
  </jaxrs:server>

  <bean id="enrichmentAnalysisResource"
      class="restservices.EnrichmentAnalysisResource" />
</beans>
```

Nel file web.xml si può notare come venga impostato come parametro di contesto il secondo file XML (beans.xml) nel tag <context-param>; viene quindi attivato il listener fornito dal framework Spring; viene registrata la CXFServlet, una servlet che si attiva all'atto del deployment su server e che riceve le richieste dirette ai servizi. Si può notare che tale servlet viene mappata sul percorso "/services/*", cioè tutti i servizi sviluppati saranno raggiungibili all'interno di tale directory.

Nel file beans.xml vengono importati tutti i file di configurazione di CXF e viene dato un nome alla risorsa che fornisce i servizi ("enrichmentAnalysisService"); tale risorsa viene mappata sulla classe Java e in seguito ad un indirizzo: nel codice sviluppato è stato impostato come indirizzo "/", quindi i servizi sono raggiungibili all'interno della directory "/services" indicata in precedenza (se fosse stato specificato un indirizzo diverso i servizi sarebbero stati raggiungibili in una sotto-directory di "/services").

6.3.7 Gestione delle sessioni

Un aspetto critico del sistema realizzato è la gestione delle sessioni. Infatti, mentre nello sviluppo di una classica applicazione web è possibile sfruttare il meccanismo di sessioni offerto da JSP e servlet, basandosi su servizi web RESTful, per definizione stateless, non è possibile avvalersi di sessioni a livello applicativo. Poiché i servizi sviluppati potranno essere utilizzabili anche da altri sviluppatori software per essere integrati in processi più complessi, ma richiedono comunque un'autenticazione, la gestione delle sessioni è stata implementata a livello di database.

6.3.7.1 Gestione delle sessioni in ambito REST

È stata quindi creata una struttura per la memorizzazione dell'ultima sessione di un utente all'interno del database del sistema (come visibile in figura 6.4, entità LAST SESSION); questa struttura viene popolata da funzioni integrate all'interno dei servizi: il servizio di login inserirà nella struttura il timestamp di inizio sessione, il servizio di logout il timestamp di fine, mentre le altre operazioni aggiorneranno i timestamp di ultima azione e modifica. Quindi un utente (o un sistema automatico) che invoca i servizi sviluppati dovrà necessariamente autenticarsi, e in seguito potrà usufruire di una sessione di una certa durata stabilita, proprio come avviene per le normali applicazioni web.

L'utilizzo di sessioni è necessario poiché il sistema ha la necessità di mantenere temporaneamente alcune informazioni in tabelle del database. Poiché le tabelle temporanee disponibili in PostgreSQL rimangono valide e accessibili solo all'interno di una stessa connessione, laddove viene aperta una nuova connessione a ogni invocazione di un servizio stateless, si è deciso di memorizzare i dati temporanei in normali tabelle persistenti eliminate al logout dell'utente. Un problema in questo meccanismo è dovuto alla scadenza delle sessioni. Se un utente non effettua il logout, non sarà possibile per il sistema cancellare le tabelle con i dati temporanei; è stata quindi realizzata una procedura per l'eliminazione delle tabelle di dati temporanei create da un utente che si attiva al login e rimane attiva in background. La procedura controlla la tabella delle sessioni nel database: se un utente non ha una sessione in corso ed è presente un timestamp di fine sessione (quindi è il caso di sessione chiusa correttamente) la procedura termina; se l'utente ha una sessione in corso la procedura si mette in attesa e rieffettua il controllo dopo che è trascorso un certo intervallo di tempo; se la sessione dell'utente è scaduta la procedura si attiva e cancella le tabelle contenenti i dati temporanei relative al dato utente, quindi termina.

6.4 Gestione dell'aggiornamento della base di conoscenza

6.4.1 Definizione del problema

I sistemi informativi web si avvalgono sempre più spesso di basi di conoscenza per la gestione e l'integrazione di dati. Tali basi di conoscenza possono integrare dati omogenei e provenienti dalla stessa sorgente, oppure dati provenienti da sorgenti diverse appartenenti ad enti diversi (aziende, università, fondazioni). Questo approccio è notevolmente diffuso nell'ambito bioinformatico: esistono numerose banche dati mondiali contenenti informazioni di tipo biomedico e biomolecolare i cui dati vengono integrati in grandi basi di conoscenza che sono alla base di strumenti per l'analisi di dati sperimentali e la generazione di nuove informazioni e conoscenza. Se la base di conoscenza considerata integra dati di sorgenti che variano nel tempo, essa dovrà a sua volta aggiornare i dati integrati: si parla in questo caso di base di conoscenza in evoluzione. L'aggiornamento di una base di conoscenza in evoluzione può essere di due tipi: incrementale, se i dati aggiornati si aggiungono a quelli già presenti; sostitutivo, se i dati integrati devono essere ricalcolati a ogni importazione dalle sorgenti esterne. Questa ultima tipologia di aggiornamento, frequente nell'ambito bioinformatico, implica la creazione di una nuova versione della base di conoscenza (verosimilmente su una macchina fisica diversa, per non limitare le prestazioni di quella attiva); in seguito sarà attivata la nuova versione e disattivata quella precedente.

Nel caso un sistema informativo sfrutti una base di conoscenza incrementale, i dati correlati a essa saranno consistenti anche dopo un'evoluzione della stessa: infatti i dati della base di conoscenza non vengono sostituiti, ma solo ampliati. Un sistema informativo che sfrutti una base di conoscenza non incrementale invece dovrà tenere conto delle problematiche relative alla sua evoluzione: sarà necessario conservare la correlazione esistente fra i dati del sistema informativo e quelli della base di conoscenza durante l'evoluzione e allo stesso tempo mantenere l'operatività del sistema il più a lungo possibile. Il problema dell'operatività si pone se i dati generati dal sistema informativo ed interagenti con la base di conoscenza sono numerosi, poiché il processo per garantire la conservazione della correlazione tra i dati può essere molto lungo.

I dati di un sistema informativo potrebbero essere memorizzati in un database proprio, oppure in uno schema all'interno della base di conoscenza; ciò dipende dalle scelte di progettazione di tale sistema e dal supporto del DBMS utilizzato alle query fra database diversi (considerando anche la possibilità che i database risiedano su macchine fisiche differenti). Se il DBMS permette di effettuare query fra database su macchine diverse, mantenere la correlazione fra i dati di un sistema informativo e della base di conoscenza sfruttata non presenta particolari difficoltà; l'efficienza di questa soluzione si scontra però con un altro aspetto dei sistemi data intensive: l'enorme quantità di dati generata e gestita da tali sistemi. Infatti, l'esecuzione di query che coinvolgono anche centinaia di migliaia di tuple fra database su macchine diverse comporta un calo di prestazioni non indifferente. L'alternativa migliore, quindi, è l'utilizzo di schemi di dati del sistema all'interno della base di conoscenza oppure l'utilizzo di database che risiedano sulla stessa macchina di tale base di conoscenza.

Questa alternativa richiede però un'attenzione particolare durante la fase di evoluzione della base di conoscenza. Infatti quest'ultima viene ricreata da zero, ed è necessario riportare sulla stessa macchina che la contiene una nuova versione di tutti i dati memorizzati precedentemente dal sistema informativo, evitando perdite di informazioni e garantendo il più possibile l'operatività di tale sistema.

Nei paragrafi seguenti verranno illustrate alcune proposte studiate per garantire tali requisiti.

6.4.2 Proposta per la risoluzione del problema

Si suppone di avere a disposizione un database, per esempio chiamato KB_system (il cui schema è illustrato in figura 6.6), che contenga informazioni generali sulla base di conoscenza, e in particolare informazioni sui suoi aggiornamenti (timestamp di inizio e fine aggiornamento della base di conoscenza e di ogni schema e database dei sistemi informativi o delle applicazioni che utilizzano la base di conoscenza). Tali informazioni saranno necessarie per rilevare evoluzioni della base di conoscenza e per gestire il processo di aggiornamento dei dati dei sistemi informativi che la sfruttano (processo indispensabile per mantenere la correlazione fra i dati della base di conoscenza e quelli dei sistemi appoggiati a essa).

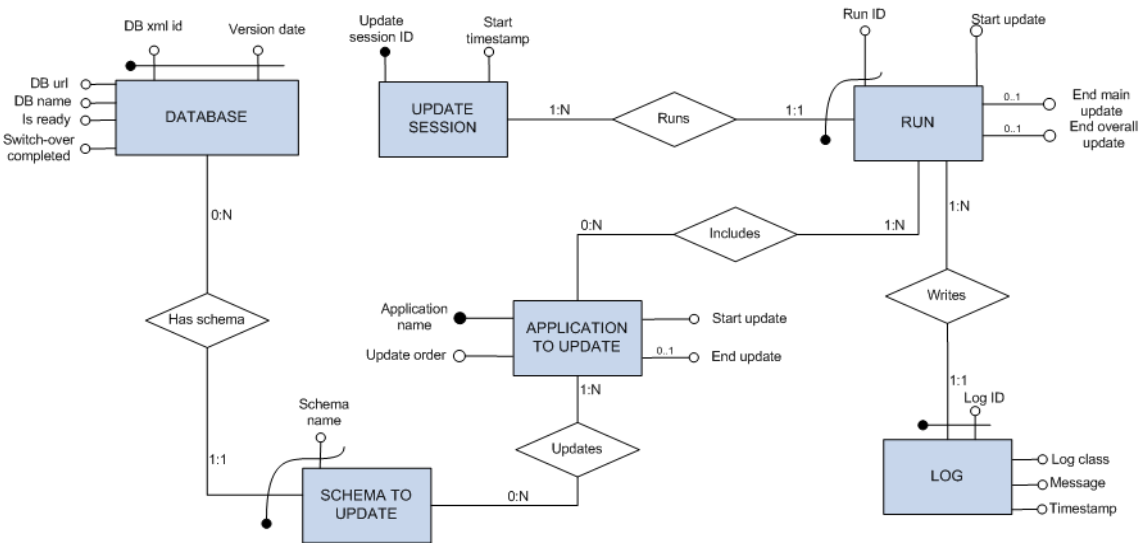


Figura 6.6: Schema del database KB_system

L'entità UPDATE SESSION rappresenta un singolo aggiornamento della base di conoscenza e dei database dei sistemi costruiti su essa; l'entità RUN rappresenta l'aggiornamento di tutti i dati di un generico sistema informativo che si appoggia alla base di conoscenza (o di gruppi di sistemi). Sono necessarie alcune osservazioni anche sull'entità DATABASE: l'attributo *Is ready* indica se il database di un sistema è pronto per cominciare un aggiornamento dei propri dati, *Switch-over completed* indica che la tale procedura di aggiornamento è terminata ed è possibile usare di nuovo tutti i sistemi poggiati sulla knowledge base.

In seguito la procedura di aggiornamento dei dati di un sistema informativo basato su una base di conoscenza verrà chiamata "procedura di switch-over".

Si possono identificare diversi stati per la procedura di switch-over:

1. stato precedente all'inizio del switch-over
2. stato di inizio switch-over
3. aggiornamento principale in corso (aggiornamento di tutte le tabelle nella nuova versione degli schemi dei dati del sistema informativo)
4. aggiornamento dei dati presenti negli schemi del sistema modificati dagli utenti durante il switch-over
5. stato di fine switch-over.

In particolare, nello stato di inizio switch-over, si controlla che nel database contenente la base di conoscenza siano presenti alcuni schemi e tabelle base indicati in un file di configurazione, necessari al funzionamento di tale base di conoscenza. Per modellizzare una soluzione al problema, è necessario tenere conto della tipologia di accesso degli

utenti al sistema informativo web: la schematizzazione delle varie tipologie è mostrata in figura 6.7. Si consideri che in figura vengano indicati gli istanti corrispondenti alla prima e all'ultima modifica dei dati da parte degli utenti (l'intervallo di tempo racchiuso fra questi due istanti verrà chiamato nei paragrafi seguenti "Sessione di lavoro", diverso dalla sessione di un utente nel sistema informativo, che può avvenire senza contemplare la modifica dei dati).

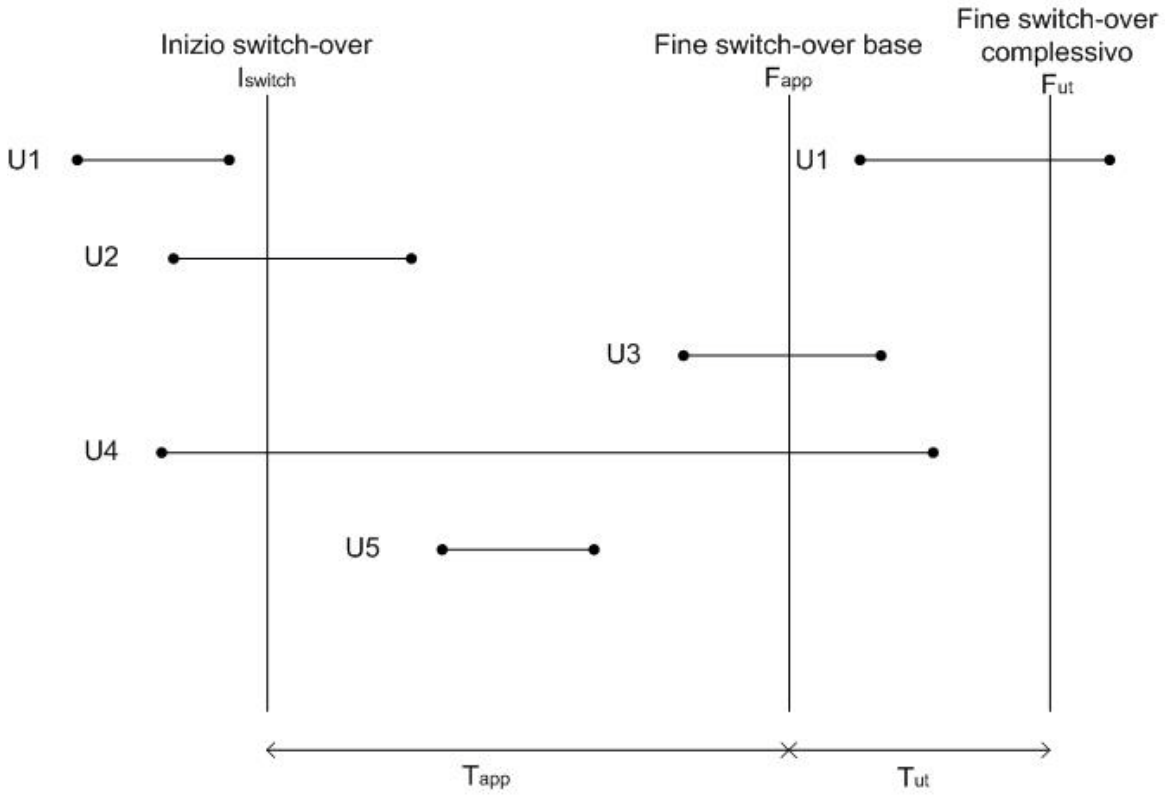


Figura 6.7: schema dei comportamenti possibili degli utenti durante l'aggiornamento del database; Fapp indica la fine dell'aggiornamento di tutte le tabelle di un sistema informativo; Fut indica la fine del riaggiornamento delle tabelle contenenti i dati modificati dagli utenti durante la prima parte del switch-over

Nell'intervallo fra I_{switch} e F_{app} la procedura di aggiornamento esegue la copia nella nuova versione di database di tutte le tabelle degli schemi di dati del sistema informativo presenti nella precedente versione. Questa copia richiede un tempo T_{app} piuttosto elevato se la quantità di dati presenti è cospicua.

Nell'intervallo successivo, T_{ut}, la procedura di aggiornamento dovrà eseguire di nuovo la copia dei dati presenti negli schemi del sistema modificati dagli utenti durante l'intervallo di tempo T_{app}.

Si possono distinguere tre casi:

1. ogni utente ha negli schemi del database un proprio insieme di tabelle indipendenti da quelle degli altri utenti
2. le tabelle sono comuni a più utenti, ma essi non condividono l'accesso in modifica dati
3. i dati di utenti diversi sono dipendenti fra loro: si tratta di dati su cui agiscono più utenti o di dati che vengono influenzati da quelli di altri utenti.

I primi due casi verranno trattati congiuntamente nel capitolo 6.4.3, considerato che il caso di tabelle indipendenti è un aspetto limite del caso di dati indipendenti; il terzo caso verrà trattato brevemente nel capitolo 6.4.4.

Tutte le soluzioni al problema proposte nei paragrafi seguenti sono indipendenti dal DBMS utilizzato, quindi valide in generale.

6.4.3 Caso di utenti con dati indipendenti

Verrà illustrato per prima cosa un procedimento applicabile in maniera generale per il caso in cui gli utenti di un sistema informativo web possiedono dati indipendenti l'uno dall'altro, imponendo al sistema dei requisiti minimi per la riuscita dell'aggiornamento degli schemi.

Per illustrare lo svolgimento della procedura di switch-over si farà riferimento alle tempistiche di accesso degli utenti indicate in figura 6.7.

Tutti gli utenti agiscono sulla vecchia versione dei dati fino all'istante F_{app}; chi si connette dopo F_{app}, non avendo acceduto in modifica durante l'intervallo T_{app}, potrà già utilizzare la nuova versione. Inoltre, per sapere a quale versione del data warehouse contenente la base di conoscenza appoggiarsi, all'inizio di ogni sessione il sistema dovrà verificare quale è attivo al momento e se è in corso un aggiornamento (tutte informazioni reperibili nella tabella *database* di KB_system). Durante l'intervallo T_{ut} vengono ripristinate le tabelle di un utente per volta (ciò non genera errori dato che gli utenti hanno dati indipendenti fra loro). Seguendo questo procedimento, un utente i cui dati sono già stati ripristinati completamente nella nuova versione del data warehouse, può accedere al sistema senza attendere la fine complessiva del switch-over.

Si consideri l'utente U1, che inizia e termina la sua sessione di lavoro prima dell'inizio del switch-over oppure inizia la sessione di lavoro dopo F_{app} (istante di fine copia di tutti gli schemi del sistema presenti nella base di conoscenza) e non accede in modifica durante T_{app}: per questa tipologia di utente non vi sono problemi, egli potrà sempre

lavorare sulla versione dei dati più aggiornata, poiché, non avendo modificato i propri dati mentre era in corso un aggiornamento, non sarà necessario ripristinarne la versione più recente.

Per quanto riguarda gli utenti delle tipologie U2 e U5, che terminano la sessione di lavoro durante la prima parte del switch-over, si presenta un problema: infatti nell'intervallo fra F_{app} e F_{ut} i loro dati dovranno di nuovo essere ripristinati nella nuova versione della base di conoscenza a causa delle modifiche effettuate durante T_{app}. Quindi tali utenti, nel caso volessero effettuare una nuova sessione di lavoro in questo lasso di tempo, non potranno accedere al sistema informativo fino all'istante di terminazione complessiva del switch-over (T_{ut}). La gestione della copia nella nuova versione di database dei dati modificati comincia all'istante F_{app} per tutti gli utenti di questa tipologia.

Le problematiche relative agli utenti U2 e U5 si presentano anche per gli utenti U3 e U4, con la differenza che la gestione della copia dei dati modificati da tali utenti durante l'intervallo di tempo T_{app} inizia dopo la terminazione della loro sessione di lavoro.

È possibile progettare diverse soluzioni al problema dell'aggiornamento di schemi di dati di un sistema informativo dipendenti dai dati presenti all'interno della base di conoscenza in base ai requisiti richiesti al sistema che la utilizza. In tabella 6.1 sono illustrati i requisiti minimi richiesti al sistema e i requisiti per sviluppare una soluzione ottimizzata.

Requisiti minimi	Il sistema deve esporre la tabella utenti e in particolare un campo di abilitazione, che identifichi se l'utente può accedere o meno. Inoltre deve essere garantito alla procedura di ripristino l'accesso in lettura alla tabella delle sessioni.
Requisiti per soluzione ottimizzata	Il sistema deve esporre la tabella utenti e in particolare un campo di abilitazione e un campo che indichi se l'utente va messo in coda per il ripristino di qualche sua tabella. Inoltre devono essere accessibili dalla procedura di ripristino anche le tabelle che contengono le informazioni sui dati da ripristinare e la lista di operazioni da eseguire.

Tabella 6.2. Requisiti richiesti al sistema per l'aggiornamento

6.4.3.1 Soluzione a requisiti minimi

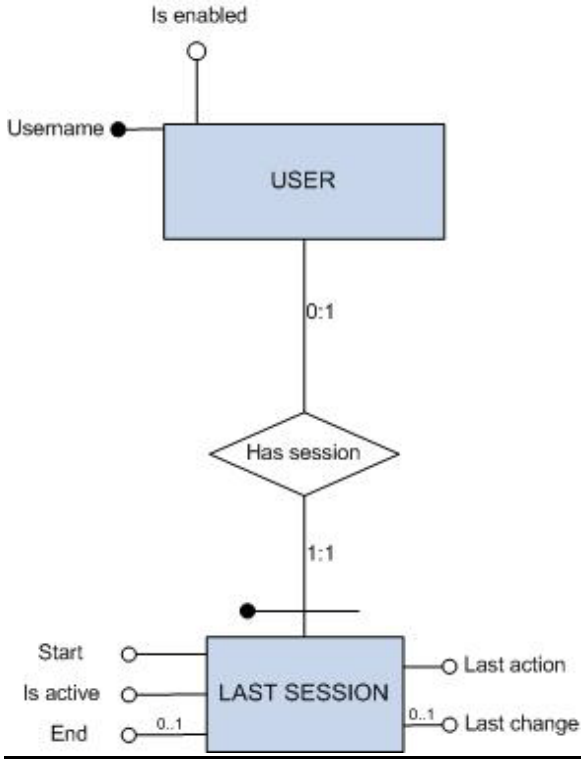


Figura 6.8: Schema minimo esposto del database di un sistema che permetta il ripristino tramite la procedura studiata

Considerando il caso più generale possibile, in cui un sistema informativo non offre alla procedura di aggiornamento alcun accesso in scrittura ai suoi dati al di fuori della tabella utenti contenente un campo di abilitazione, l'unica soluzione è imporre un tempo di attesa fra gli istanti F_{app} e F_{ut} per tutti gli utenti che hanno effettuato una sessione di lavoro durante T_{app} e che vogliono iniziarne una nuova durante l'intervallo T_{ut}, in modo che nessun utente possa modificare dei dati mentre vengono esportati nella nuova versione di database. È la procedura di ripristino a verificare se gli utenti hanno modificato i propri dati durante la prima fase dell'aggiornamento sfruttando la tabella delle sessioni e le informazioni riportate nel database KB_system, e ad abilitare o disabilitare l'utente. Seguendo il diagramma in figura 6.7, l'utente U1 sarebbe sempre abilitato, mentre gli altri verrebbero disabilitati all'istante F_{app} e riabilitati all'istante F_{ut} (salvo il caso in cui la sessione di lavoro dell'utente termina durante T_{ut}). Considerando che gli utenti che accederanno in modifica al database durante la fase di aggiornamento verosimilmente saranno una minima parte del totale, questo intervallo di tempo sarà quindi piuttosto ridotto rispetto a T_{app}.

6.4.3.2 Soluzione ottimizzata

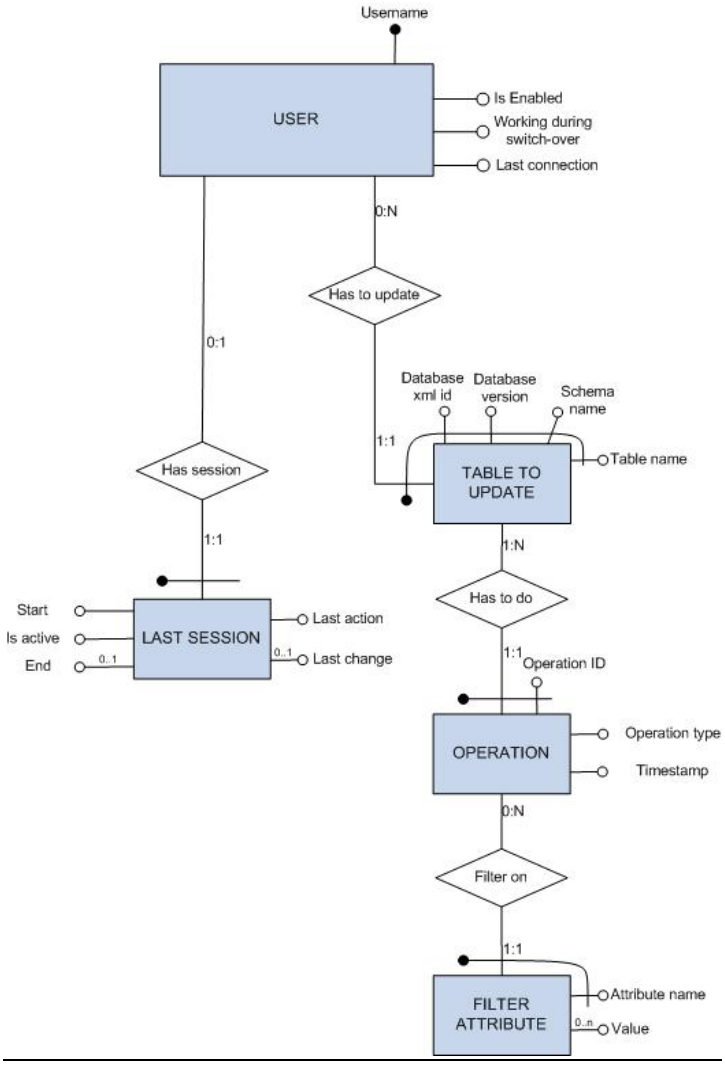


Figura 6.9: Schema E-R del modello di database esposto che permette il ripristino tramite la soluzione ottimizzata considerata

A questo punto si può considerare una soluzione ottimizzata, che riduce mediamente i tempi per il ripristino dei dati degli utenti, ma che richiede l'esposizione di maggiori informazioni sull'attività degli utenti da parte del sistema informativo considerato. È necessario ipotizzare che il sistema debba salvare in una tabella di un proprio database separato e stabile le informazioni relative ai dati modificati dall'utente durante l'aggiornamento: queste informazioni saranno dei "filtri", che permettono di identificare sottoinsiemi di tuple da ripristinare per ogni tabella (o tabelle intere), in modo da ridurre al minimo i dati da aggiornare nuovamente nell'intervallo T_{ut}; tali informazioni verranno successivamente recuperate dalla procedura di aggiornamento e utilizzate per il ripristino selettivo dei dati modificati.

Si ipotizzi che il sistema conceda l'utilizzo di una tabella con elenco degli utenti contenente un campo *Is enabled* e un campo *Working during switch-over*, l'accesso in lettura alla tabella delle sessioni contenente, per ogni utente, il timestamp di inizio e fine sessione, il timestamp dell'ultima azione compiuta e dell'ultima modifica ai dati, un campo che indica se l'utente ha in corso una sessione o meno, e l'accesso completo alle tabelle con le informazioni sui dati modificati da ripristinare (come visibile in fig. 6.9). In questo caso: tutti gli utenti che hanno in corso una sessione di lavoro durante l'intervallo T_{app} vengono impostati come attivi durante il switch-over (attributo *Working during switch-over* vero) all'istante F_{app}. Vengono considerati anche tutti gli utenti che ancora non hanno terminato la sessione di lavoro perché, nel caso pessimo, potrebbe capitare che un utente svolga una sessione molto lunga, tale da terminare dopo che il ripristino di tutti i dati degli altri utenti sia già concluso: in questo caso, se l'utente in questione non fosse segnalato come detto, la procedura di aggiornamento avrebbe termine prima di poter ripristinare i suoi dati alla versione più recente. A questo punto la procedura di aggiornamento può creare una coda FIFO a partire dai dati ricavati dalla tabella utenti, da quella delle sessioni e dal database KB_system: vengono aggiunti in coda per il ripristino dei propri dati solo gli utenti che non abbiano una sessione in corso, e si ripristinano i dati di un utente per volta; quando i dati del suddetto utente sono stati ripristinati, lo si imposta come abilitato nella tabella degli utenti, lo si elimina dalla coda e si imposta a falso l'attributo *Working during switch-over*. Inoltre vengono rimossi tutti i dati relativi alle tuple ripristinate durante l'aggiornamento. Ad ogni avvio della sessione, il sistema controlla il campo di abilitazione, così che un utente possa accedere al sistema anche durante l'intervallo critico T_{ut}, a patto che sia stato abilitato. Queste piccole modifiche permettono di ridurre mediamente il tempo di attesa per degli utenti che vogliono connettersi prima della fine del switch-over complessivo: infatti non tutti gli utenti dovranno attendere durante l'intervallo T_{ut}, ma solo coloro per cui il ripristino dei dati modificati non è ancora stato effettuato. La procedura di ripristino termina quando nessun utente ha più il campo *Working during switch-over* settato a vero.

Può capitare però che l'ordine dei dati da ripristinare preveda prima la copia di dati di utenti disconnessi, e per ultima la copia di dati di utenti che tentano di connettersi durante l'intervallo T_{ut}. Per migliorare la qualità del servizio in questo caso, si può proporre un'ulteriore ottimizzazione della procedura:

- la procedura dovrà ricalcolare la coda dopo ogni iterazione del ripristino

- se un utente ha effettuato un tentativo di connessione durante l'intervallo T_{ut} (circostanza rilevabile leggendo l'attributo *Last connection* della tabella utenti), si pone in testa alla coda.

Così facendo, si dà precedenza agli utenti che tentano di connettersi in quell'intervallo a discapito degli utenti inattivi.

Un'ulteriore modifica al procedimento in favore della diminuzione dei tempi di attesa è la possibilità di effettuare il ripristino in parallelo di dati di più utenti. Si possono lanciare un numero N di thread paralleli che eseguono il ripristino di dati di un utente ciascuno; il numero N ovviamente deve essere limitato, dato che il collo di bottiglia delle operazioni sono le scritture sui dischi fissi, e un numero di thread troppo elevato pregiudicherebbe i vantaggi ottenuti. Non vi sono problemi nel ripristinare dati di più utenti in parallelo, poiché i dati considerati sono indipendenti fra loro per ipotesi. Se un utente effettua un tentativo di connessione mentre è in corso il ripristino di dati di altri utenti, è possibile fare in modo che sia il sistema a lanciare un thread per il ripristino dei dati dell'utente in questione: così l'utente dovrebbe solo attendere il tempo necessario alla copia dei suoi dati. Questa modifica però richiede requisiti maggiori: infatti è necessario che sia il sistema informativo stesso a poter lanciare l'esecuzione della procedura di ripristino nell'istante del tentativo di connessione dell'utente.

6.4.3.3 Gestione degli errori

È necessaria ovviamente anche una gestione degli errori per la procedura di switch-over. Vengono considerati errori imputabili a cause esterne (ad esempio interruzioni dell'alimentazione, interruzioni della rete, crash dei processi); non vengono considerati errori dovuti ai dati, poiché, essendo memorizzati nella precedente versione del database è certo che non vi siano errori in essi (altrimenti il precedente database non avrebbe potuto essere compilato). Si suppone che la procedura di aggiornamento dei dati dei vari sistemi si avvii dopo che la procedura di aggiornamento degli schemi della base di conoscenza abbia avuto successo. Vengono ripristinati tutti gli schemi e i database dei sistemi informativi, uno per volta (o a gruppi, se i loro dati sono tra loro dipendenti): questa operazione verrà chiamata "run" di aggiornamento. Si considera che ogni "run" sia indipendente da ogni altro. Se avviene un errore nella fase principale di switch-over, il procedimento riparte da capo, senza ulteriori conseguenze, se non una perdita di tempo. Se l'errore avviene nella fase di copia dei dati modificati dagli utenti durante

Tapp (quindi la fase di aggiornamento principale si è conclusa con successo), è sufficiente ricominciare il "run" di aggiornamento per il determinato utente per cui è avvenuto l'errore (dato che i dati degli utenti sono indipendenti, e vengono ripristinati sequenzialmente tabella per tabella). Poiché vi sono degli utenti in coda, però, è auspicabile un numero massimo basso di tentativi di ripristino: superato questo numero, l'aggiornamento dei dati di un determinato utente viene temporaneamente sospeso e l'utente viene posizionato in fondo alla coda. Se in un dato istante rimangono in coda solo degli utenti per cui durante l'aggiornamento dei dati continuano a verificarsi errori, si procede di nuovo per un certo numero N grande di tentativi. Se si presentano degli errori superato questo numero N di tentativi, vi sono due soluzioni: se i dati non sono critici, e una modifica dell'utente ai dati non aggiornati non causa danni, si abilita comunque l'utente avvisandolo che lo schema del data warehouse contiene dati non aggiornati, e inoltre si avvisa l'amministratore di sistema (con una entry nella tabella di log) che potrà tentare il ripristino manuale in seguito; se i dati sono critici, invece, è necessario lasciare l'utente disabilitato per prevenire danni irreversibili e avvisarlo dell'errore, oltre, ovviamente, ad avvisare l'amministratore di sistema con una entry nella tabella di log. La scelta fra queste due opzioni può essere demandata a un file di configurazione. Un caso critico si presenta quando avviene un errore durante la modifica delle tabelle di sistema relative all'aggiornamento o degli attributi *Is enabled* e *Working during switch-over* della tabella utenti: una soluzione a questo problema è trattare questa parte come una transazione separata, cosicché il fallimento non porti a dover ricominciare il "run" di aggiornamento, ma solo a dover modificare di nuovo le suddette tabelle. Si ha comunque la garanzia che, se la procedura di switch-over è giunta a modificare le tabelle di sistema, l'aggiornamento dei dati è andato a buon fine.

6.4.4 Caso di utenti con dati dipendenti

Il caso di aggiornamento di schemi con dati dipendenti fra più utenti è molto complesso, e non sarà trattato approfonditamente in questo lavoro di Tesi, ma solo introdotto brevemente.

I casi in cui vi sono dati dipendenti fra più utenti in realtà sono due:

- gli utenti agiscono in modo collaborativo sugli stessi dati
- i dati di un utente presenti in una certa tabella dipendono da dati di altri utenti presenti in altre tabelle (quindi sono vincolati da chiavi esterne nel database).

In questi casi, l'aggiornamento del database deve presupporre che vi sia un certo ordinamento fra le modifiche ai dati da parte del sistema, e che non vi siano due o più utenti che agiscano contemporaneamente sulle stesse tuple; quindi i dati devono essere sempre consistenti. Questo requisito è soddisfatto dal DBMS, quindi si può passare a discutere le modalità di aggiornamento. Per quanto riguarda la fase principale di aggiornamento (la copia di tutte le tabelle degli schemi selezionati), essa avviene in maniera analoga al caso di dati indipendenti, poiché la procedura copia tutte le tabelle seguendo l'ordine delle loro dipendenze. Cambia invece la fase di ripristino dei dati modificati durante l'aggiornamento da parte degli utenti.

Se si agisce su dati comuni in maniera collaborativa, è sufficiente che sia indicata alla procedura di aggiornamento l'ultima modifica su essi: ciò implica che ogni utente che li condivide debba attendere il termine del loro aggiornamento. Il problema è molto complesso, soprattutto nel caso si vogliano limitare i tempi di attesa per gli utenti: infatti considerando il caso base, se l'ultima modifica su un certo dato è nella coda di ripristino di un utente A, ma tale dato è condiviso dall'utente B, l'utente B dovrà attendere fino alla fine del ripristino dei dati di A per accedervi. È quindi necessario studiare un approccio completamente diverso per attuare delle ottimizzazioni al procedimento. Se si parla di dati dipendenti fra tabelle di utenti diversi, invece, è necessario, all'atto dell'aggiornamento, avere una lista di tutte le tabelle da aggiornare e trovare le dipendenze richieste per ogni tabella in esame: vengono quindi aggiornati i dati seguendo l'ordine delle dipendenze; non è più possibile aggiornare i dati di ogni utente separatamente. Questo contribuisce a prolungare l'intervallo di attesa per gli utenti in coda.

6.4.4.1 Gestione degli errori

Per quanto riguarda la procedura di aggiornamento principale (il ripristino di tutte le tabelle del sistema), si gestiscono gli errori come nel caso di utenti con dati indipendenti. La gestione errori della parte di ripristino dei dati modificati è invece molto più complessa nel caso di dati comuni fra più utenti, e non verrà trattato per esteso in questa Tesi; verranno trattati solo due casi base validi se il ripristino non avviene utente per utente.

Nel caso in cui le tabelle possano essere ripristinate in un ordine qualsiasi, è sufficiente, se si verifica un errore, ricominciare da zero la procedura di aggiornamento delle tabelle

modificate durante il switch-over. Altrimenti sarebbe possibile migliorare la procedura in questo modo: per evitare la riscrittura di dati di molte tabelle, gestire il ripristino di una tabella per volta seguendo le dipendenze, e all'insorgere degli errori, ricopiare solo la tabella che li ha generati. Ovviamente, nel caso il problema non si risolva al successivo tentativo, è necessario ricominciare da zero il ripristino; non è possibile mettere in coda la tabella in questione poiché da essa possono dipendere i dati presenti in altre.

6.4.5 Soluzione implementata

Per quanto riguarda il sistema sviluppato in questa Tesi, ci si pone nel caso di dati indipendenti (in scrittura) per utenti diversi (per l'esattezza, alcune tabelle contengono dati di vari utenti indipendenti fra loro, mentre altre, grazie alla funzione di partizionamento delle tabelle presente in PostgreSQL, sono separate per i vari utenti, che possiedono una tabella fisica ognuno). Il sistema sviluppato rispetta tutti i requisiti citati per l'utilizzo della soluzione ottimizzata descritta nel capitolo 6.4.3.2. Lo schema della parte di database relativa alla gestione delle sessioni, dei log e delle tabelle utilizzate per memorizzare informazioni riguardanti le tabelle da ripristinare che è stato utilizzato è lo stesso illustrato in fig. 6.9. Il sistema possiede uno schema di dati all'interno della base di conoscenza che contiene i dati che devono interagire con essa.
La base di conoscenza su cui il sistema poggia è la Genomic and Proteomic Knowledge Base, contenuta nel data warehouse GPDW. Si ha inoltre a disposizione un database contenente informazioni di sistema riguardo il GPDW chiamato GPDW_system, che rispetta lo schema illustrato in fig. 6.6.
Il sistema sviluppato, ad ogni avvio di una sessione di un utente, controlla nel database GPDW_system quale versione della base di conoscenza è attiva in quel dato istante, controlla se l'utente è abilitato, e in caso affermativo si connette a essa. Viene impostato a vero l'attributo *Working during switch-over* nell'istante F_{app} per tutti gli utenti che hanno modificato i propri dati durante T_{app}. Nell'istante F_{app} vengono disabilitati (da parte della procedura di aggiornamento) tutti gli utenti di questo gruppo la cui sessione è terminata. Gli utenti che hanno ancora una sessione in corso verranno disabilitati dopo la sua terminazione.
Il sistema, ad ogni operazione di modifica dei dati nel database, controlla se è in corso un switch-over, e, in caso affermativo, inserisce tutte le informazioni necessarie a

riconoscere i dati modificati dall'utente nelle tabelle descritte dalle entità TABLE TO UPDATE, OPERATION e FILTER ATTRIBUTE visibili in figura 6.9.
Il ripristino viene eseguito secondo questo procedimento (ci si riferisca sempre alla figura 6.7 per la nomenclatura di utenti e istanti di tempo):

- all'istante I_{switch} inizia l'aggiornamento di tutti gli schemi del sistema all'interno della nuova versione della base di conoscenza; tale ripristino termina all'istante F_{app}. Gli utenti che nel frattempo si connettono o hanno una sessione in corso durante l' intervallo T_{app}, continuano ad agire sulla vecchia versione dei dati
- all'istante F_{app} parte la procedura di ripristino dei dati degli utenti al lavoro durante T_{app}: questo è possibile recuperando le informazioni dalle tabelle descritte dalle entità USER, TABLE TO UPDATE, OPERATION e FILTER ATTRIBUTE. A questo punto nella tabella USER sarà vero il campo *Working during switch-over* per tutti gli utenti menzionati; la procedura di aggiornamento comincerà a copiare le tabelle di un utente per volta
- all'istante F_{app}, inoltre, la procedura controlla quali utenti hanno effettuato delle operazioni durante l'aggiornamento, e li disabilita temporaneamente
- all'istante F_{app}, la procedura crea una coda contenente gli utenti che hanno modificato i propri dati durante l'aggiornamento e che sono disabilitati, quindi inizia il ripristino dei dati per ogni utente in coda. Non viene seguito un ordine stabilito
- se durante l'intervallo di tempo T_{ut} un altro utente termina la propria sessione, egli verrà disabilitato e quindi aggiunto alla coda per il ripristino. Sempre durante questo intervallo, se un utente già in coda effettua un nuovo tentativo di connessione, si aggiorna l'attributo "*Last connection*" nella tabella degli utenti (visibile in fig. 6.9) con il valore dell'istante corrente, e la procedura, all'iterazione successiva, controllerà questo attributo e porrà l'utente in cima alla coda. Se la stessa situazione si presenta per più utenti, la procedura comincerà il ripristino dei dati dell'utente con timestamp del tentativo di ultima connessione (avvenuta nell'intervallo a T_{ut}) antecedente a tutti gli altri
- una volta terminato il ripristino dei dati di un utente, viene settato falso l'attributo *Working during switch-over* della tabella USER e l'utente viene riabilitato settando vero l'attributo *Is enabled* della stessa tabella. Tutti gli utenti, salvo occorrenza di errori, vengono quindi riabilitati entro l'istante F_{ut}.

La gestione degli errori implementata rispecchia quanto detto nel capitolo 6.4.3.3 "*Gestione degli errori*".

6.4.5.1 Algoritmo di ripristino dei dati memorizzati

È stato sviluppato un algoritmo che permette di ripristinare tutte le tabelle di uno o più schemi da un database a un altro, utile per aggiornare tutti i dati di un sistema all'evoluzione della base di conoscenza.

La procedura di ripristino sviluppata è generica: considera per il ripristino vincoli, indici e ereditarietà fra tabelle (anche a più livelli), viste, trigger e stored procedures. Per ereditarietà si intende la capacità di una tabella (la tabella figlia) di ereditare lo schema da un'altra (che verrà chiamata tabella padre): per il caso specifico del sistema sviluppato in questa Tesi, l'ereditarietà è una caratteristica sfruttata per implementare il partizionamento fisico di tabelle. L'unico limite della procedura sviluppata è l'ipotesi che le tabelle padre non possono contenere fisicamente dei dati (come menzionato nel capitolo 4.3.1).

Di seguito è illustrata una funzione in pseudo-codice che illustra lo svolgimento dell'algoritmo di ripristino di interi schemi (le funzioni sono sottolineate, le query SQL sono indicate in maiuscolo):

```
function restoreTables(Array schemas){
      var isFather, isSon;
      var sqlString, sqlConstraint, sqlIndex;
      Array createStmArray, dataStmArray;
      Array tmpStmtArray, tmpDataArray;
      Array tmpStmtArray2, tmpDataArray2;
      Array triggers, views, storedProcedures;
      connectToDB(DB_old);
      var resultSet1=execSQLStatement(SELECT_ALL_TABLES);
      while(resultSet1.hasNext){
            var resultSetF=execSQLStatement(FIND_IF_ISFATHER);
            if(resultSetF.hasNext) isFather=true;
            var resultSetS=execSQLStatement(FIND_FATHER);
            if(resultSetS.hasNext) isSon=true;
            var resultSet2= execSQLStatement(FIND_COLUMNS);
            while(resultSet2.hasNext){
                  sqlString=insertColumnIntoSQLString();
            }
            sqlConstraint=getConstraints();
            sqlIndex=getIndexes();
            sqlTrigger=getTriggers();
            if(isSon)
                  var tableStatement=createSQLSonStatement(sqlString,
                        sqlConstraint, sqlIndex);
            else var tableStatement=createSQLStatement(sqlString,
                        sqlConstraint, sqlIndex);
            createStmArray.add(tableStatement);
```

```
            if(!isFather)
                  var dataStatement=createDataSelectionStatement();
            else var dataStatement=
                  createEmptyDataSelectionStatement();
            dataStmArray.add(dataStatement);
            triggers.add(sqlTrigger);
      }
      var resultSetV=execSQLStatement(SELECT_ALL_VIEWS);
      while(resultSetV.hasNext){
                  views.add(resultSetV.next());
      }
      var resultSetSP=execSQLStatement(SELECT_STORED_PROCEDURES);
      while(resultSetsp.hasNext){
                  storedProcedures.add(resultSetSP.next());
      }
      connectToDB(DB_new);
      for(var i in 0:createStmArray.size){
            if(!table(createStmArray[i]).inherits &&
                  !table(createStmArray[i]).hasConstraints){
                  execSQLStatement(createStmArray[i]);
                  restoreData(dataStmArray[i]);
            }
            else{
                  tmpStmtArray.add(createStmArray[i]);
                  tmpDataArray.add(dataStmArray[i]);
            }
      }
      for(var i in 0:tmpStmtArray.size){
            if(!table(tmpStmtArray[i]).inherits){
                  execSQLStatement(tmpStmtArray[i]);
                  restoreData(tmpDataArray[i]);
            }
            else{
                  tmpStmtArray2.add(tmpStmtArray[i]);
                  tmpDataArray2.add(tmpDataArray[i]);
            }
      }
      for(i in 0:tmpStmtArray2.size){
            execSQLStatement(tmpStmtArray2[i]);
            restoreData(tmpDataArray2[i]);
      }
      restoreTriggers(triggers);
      restoreViews(views);
      restoreStoredProcedures(storedProcedures);
      callCleaner();
}
```

L'algoritmo si svolge in questo modo: per prima cosa vengono estratti tutti i nomi di tabelle dato in input il nome dello schema (o degli schemi) nel database (SELECT_ALL_TABLES); poi per ogni tabella si eseguono delle query per conoscere l'oid associato (un identificativo univoco per la tabella presente nel Catalog del database), se ha dei figli o un padre (FIND_IF_ISFATHER e FIND_FATHER). A questo punto si estraggono le colonne (FIND_COLUMNS) e alla fine di ogni iterazione del ciclo, la funzione *insertIntoSQLString()* aggiunge i dati relativi alla colonna a uno Statement SQL che poi verrà eseguito per creare la tabella nel nuovo database. Alla fine

del ciclo sulle colonne, tramite le funzioni *getConstraints()* e *getIndexes()* si ricavano (con query al catalog del database) eventuali vincoli e indici presenti sulla tabella, e vengono utilizzati nello statement per la creazione della tabella nel nuovo database (tramite le funzioni *createSQLSonStatement()* e *createSQLStatement()*). Viene quindi creato lo statement in maniera diversa in base al fatto che la tabella erediti o meno lo schema da un'altra (infatti, se eredita lo schema, nello statement SQL non andranno indicati i campi della tabella); quindi viene creato un altro statement (per l'ottenimento di tutti i dati della tabella in esame) in maniera diversa in base al fatto che una tabella abbia dei figli o meno. Infatti se la tabella ha dei figli, quindi è partizionata, in essa non sono inseriti fisicamente dei dati, anche se sono visibili quelli dei figli, e quindi tali dati non vanno ripristinati nel nuovo database, pena la creazione di dati duplicati. In ogni iterazione del ciclo sulle tabelle, vengono aggiunti i due statement a due array. Vengono anche recuperate informazioni relative a viste, trigger e stored procedure presenti. A questo punto si effettua una connessione al nuovo database, e scorrendo gli array vengono ripristinate le tabelle seguendo un ordine di precedenza: prima le tabelle semplici (che non ereditano e non hanno riferimenti ad altre tabelle), poi quelle con dei vincoli di tipo FOREIGN KEY, e infine quelle che ereditano. Dopo il ripristino delle tabelle vengono quindi ripristinati trigger, viste, e stored procedure tramite le funzioni *restoreTriggers()*, *restoreViews()* e *restoreStoredProcedures()*. La funzione *restoreData()* inserisce tutti i dati estratti dalla vecchia tabella nella sua nuova versione; *table()* restituisce la tabella per cui verrà eseguito uno statement SQL di creazione. Dopo il ripristino di tutte le tabelle nel database aggiornato, alla fine del procedimento, viene chiamata la funzione *callCleaner()*, che richiama alcune stored procedure utili a marcare come obsoleti alcuni dati che dipendono dall'aggiornamento della base di conoscenza, in modo da essere riconosciuti dal sistema successivamente e riaggiornati al primo utilizzo. Questa funzione dovrà essere implementata diversamente per ogni singolo sistema sviluppato sulla base di conoscenza, poiché i dati memorizzati da riaggiornare saranno differenti per ognuno.

L'algoritmo, nel caso di ripristino di sottoinsiemi di dati di singole tabelle, è leggermente diverso: vengono passati in input i nomi delle tabelle da ripristinare per ogni utente, e dei filtri che permettono di riconoscere dei sottoinsiemi di tuple; inoltre vi sono dei controlli che stabiliscono la procedura da adottare per il ripristino in base al tipo di modifica effettuata (CREATE, INSERT, DELETE, DROP, UPDATE).

Per il rilevamento delle caratteristiche delle tabelle (campi, vincoli, eventuali ereditarietà), invece, l'algoritmo è in tutto e per tutto identico al precedente.
L'algoritmo è stato poi implementato in Java basandosi su PostgreSQL come DBMS.

6.5 Uso dei servizi e dell'applicazione web sviluppati

In questo capitolo viene presentato un esempio di utilizzo del sistema tramite i servizi e l'applicazione web sviluppati: verranno illustrate le principali operazioni disponibili e gli input necessari a ognuna di esse.

6.5.1 Servizi web

I servizi web di tipo RESTful vengono invocati tramite normali richieste HTTP, per cui possono essere sfruttati da applicazioni costruite con qualunque linguaggio che permette di effettuarle. Per richiamare i servizi andranno utilizzate delle API messe a disposizione dallo specifico linguaggio per richieste HTTP; l'output dei servizi sarà un frammento di codice XML o JSON in base a quanto specificato dalla suddetta applicazione. Di seguito viene mostrato un esempio di invocazione del servizio che restituisce le sorgenti dati disponibili per una data feature biomolecolare o biomedica e un frammento di XML contenente la risposta. L'invocazione, poiché il servizio è raggiungibile tramite il metodo HTTP GET, è possibile anche da browser:

```
http://localhost:8080/GPEAer/services/gpdwservice/sources?entity=gene
```

La risposta del servizio è mostrata nel codice seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
   <Values>
       <data value="entrez_gene"/>
       <data value="ensembl"/>
       <data value="hgnc"/>
       <data value="omim"/>
   </Values>
```

Per poter utilizzare il codice XML ricevuto in risposta all'interno di un'applicazione, è necessario deserializzarlo in un oggetto del linguaggio di programmazione mediante apposite librerie (come ad esempio JAXB per Java), oppure effettuarne il parsing per estrarre le informazioni desiderate.

6.5.2 Applicazione web

Un utente, per poter utilizzare il sistema, deve necessariamente registrarsi; nella registrazione sono richieste un certo numero di informazioni obbligatorie indicate con un asterisco. Si può effettuare la registrazione come utente standard o come utente avanzato. Una volta registrato, l'utente deve autenticarsi nella schermata di login: in seguito egli avrà accesso alle funzionalità del sistema. Sono quattro le sezioni principali dell'applicazione web: il caricamento di liste di ID, la definizione di un esperimento e dei parametri di analisi, la visualizzazione dei risultati dell'analisi, la sezione di gestione dei dati salvati.

6.5.2.1 Caricamento di liste di ID

In fig. 6.10 è mostrata la schermata della pagina web in cui è possibile caricare delle liste. L'utente può decidere di caricare una o due liste per volta; deve inoltre specificate il tipo di entità biomolecolare e la sorgente dati a cui appartengono gli ID. Deve inoltre essere specificato un nome per ogni lista ed eventualmente una descrizione. Il caricamento può avvenire sia copiando la lista nell'apposita area di testo sia tramite un file di testo che la contiene. Le entità biomolecolari e le sorgenti dati sono caricate dinamicamente; in particolare la sorgente può essere scelta solo dopo aver selezionate l'entità biomolecolare. L'utente, dopo aver completato la compilazione dei campi della form può proseguire; si passa quindi ad una pagina web (in fig. 6.11) in cui si possono scegliere le colonne di classificazione eventualmente presenti da salvare e in cui vengono presentati i risultati del controllo preliminare delle liste (con segnalazione di ID obsoleti o non riconosciuti).

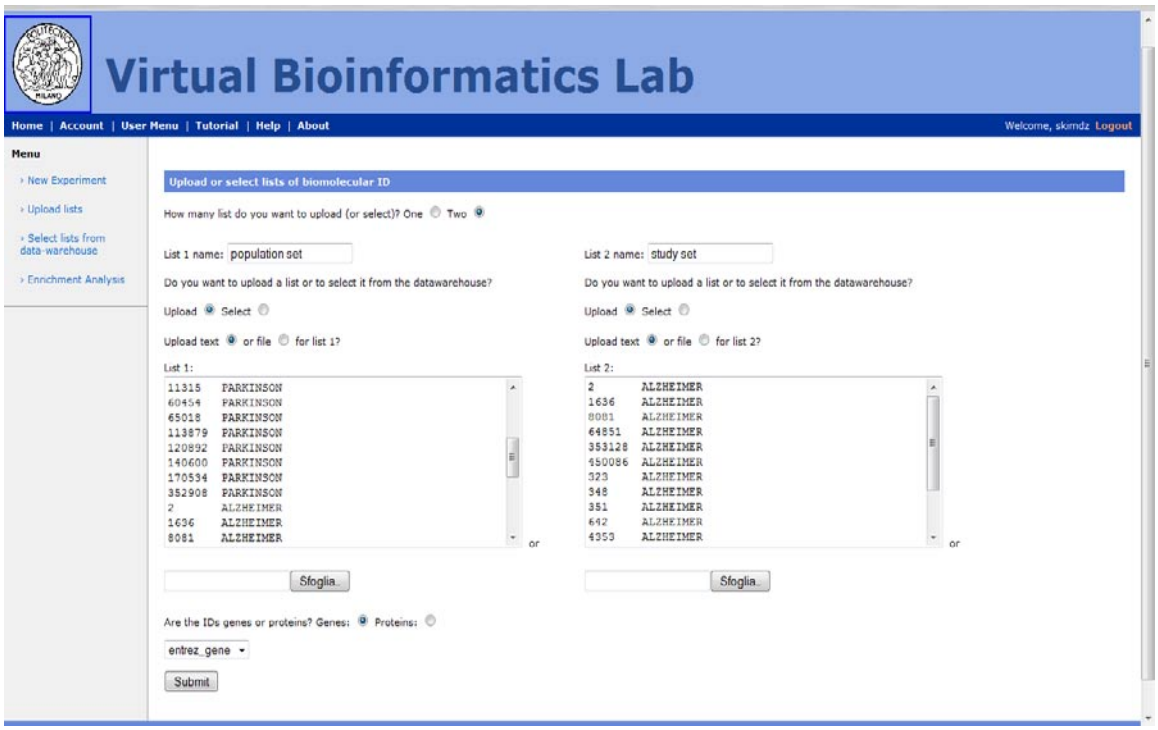


Figura 6.10. Pagina web per il caricamento di liste di ID biomolecolari

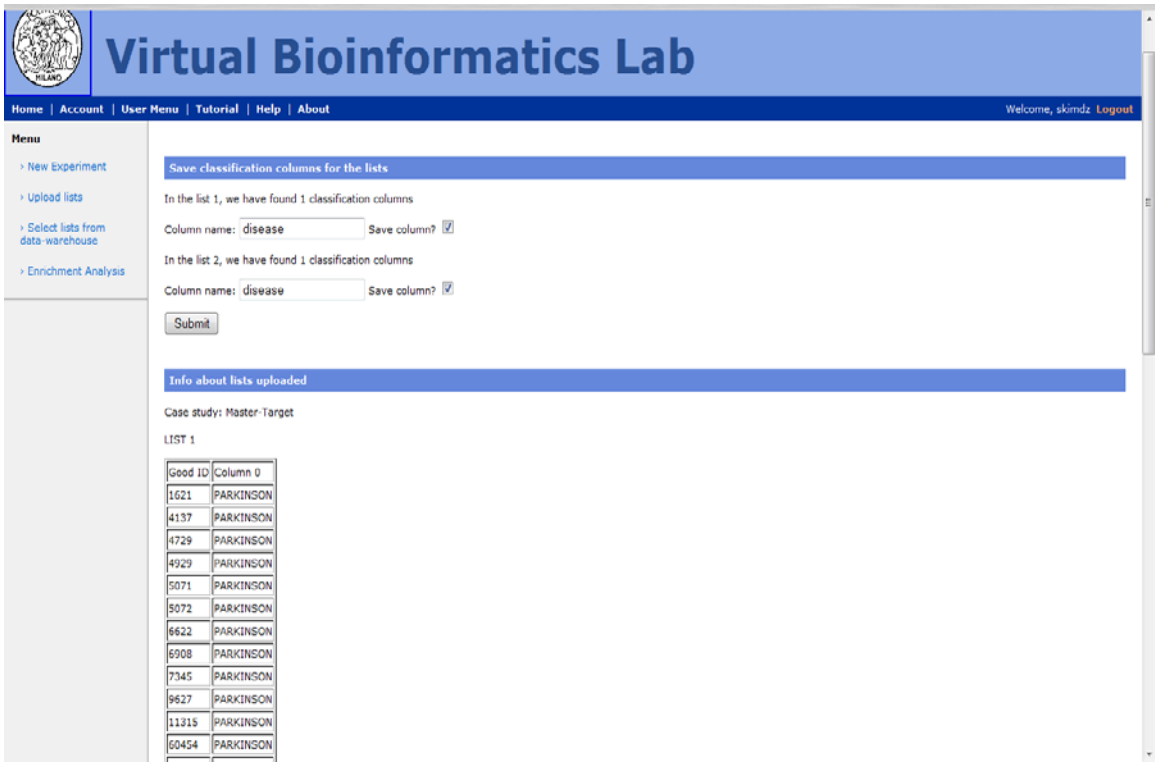


Figura 6.11. Pagina web con il salvataggio di classificazioni e informazioni sulla lista caricata

Se si prosegue, le liste verranno salvate e si giungerà ad una pagina con le informazioni risultati dalla traduzione degli ID: numero di ID riconosciuti e non riconosciuti, numero di ID effettivamente tradotti, numero di ID obsoleti.

6.5.2.2 Definizione dei parametri di analisi

In alternativa, un utente può iniziare subito con la definizione di un nuovo esperimento. Una volta creato l'esperimento (sono necessari solo un nome e una descrizione), si passa automaticamente alla pagina per il caricamento delle liste illustrata nel paragrafo precedente. Dopo aver effettuato tutte le operazioni relative, l'utente passerà a definire uno specifico sottocaso dell'esperimento, cioè la scelta di quale lista utilizzare come population set e quale come study set. Una volta stabilito, si passa ad una pagina in cui selezionare il tipo di analisi di arricchimento desiderata e i vocabolari controllati sui quali effettuarla, visibile in fig. 6.12.

Figura.6.12. Scelta delle impostazioni principali per l'analisi

È possibile scegliere un'analisi concettuale o non concettuale lato entità biomolecolare e lato feature biomedica. Le feature biomediche disponibili vengono visualizzare dinamicamente nella pagina (grazie a funzioni che sfruttano AJAX) dopo aver selezionato la tipologia di analisi per entità biomolecolare. Una volta selezionate una o più feature compariranno dinamicamente l'elenco delle sorgenti disponibili per esse e i provider di annotazioni disponibili. È possibile selezionare più feature da analizzare contemporaneamente, più sorgenti e più provider di annotazioni per ogni feature. Una

volta selezionate tutte le impostazioni desiderate si può procedere e raggiungere la pagina per la selezione dei test statistici e delle correzioni da utilizzare (fig. 6.13).

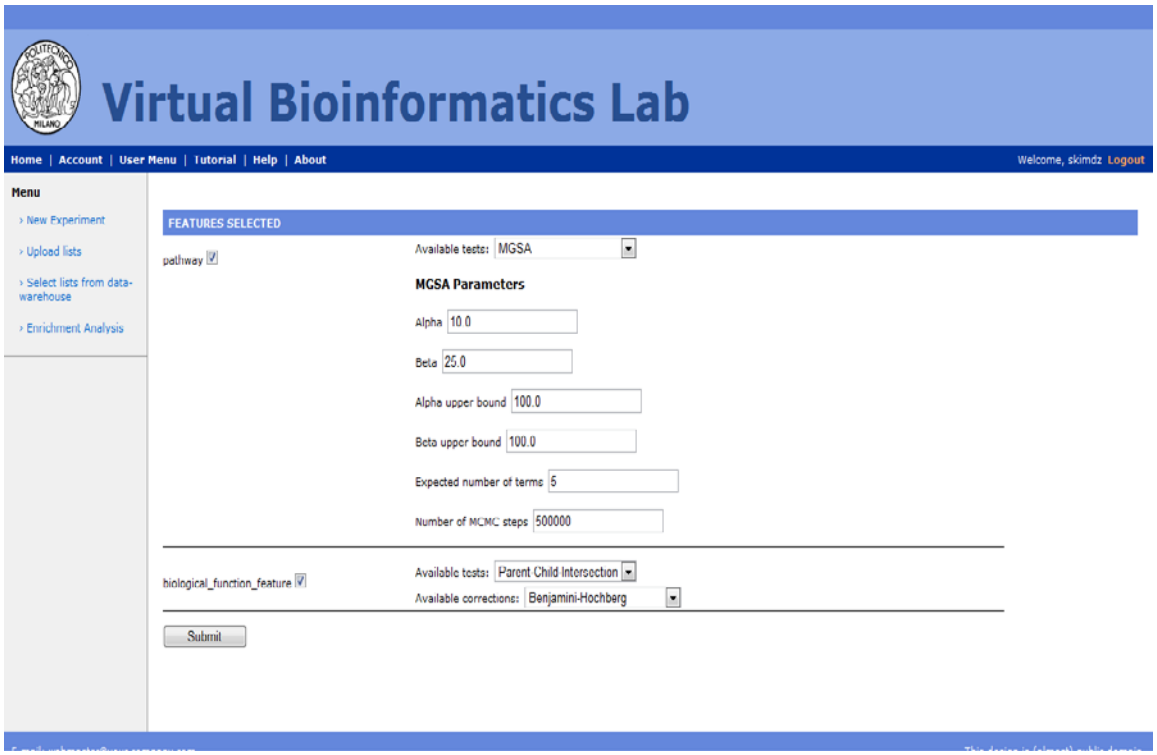
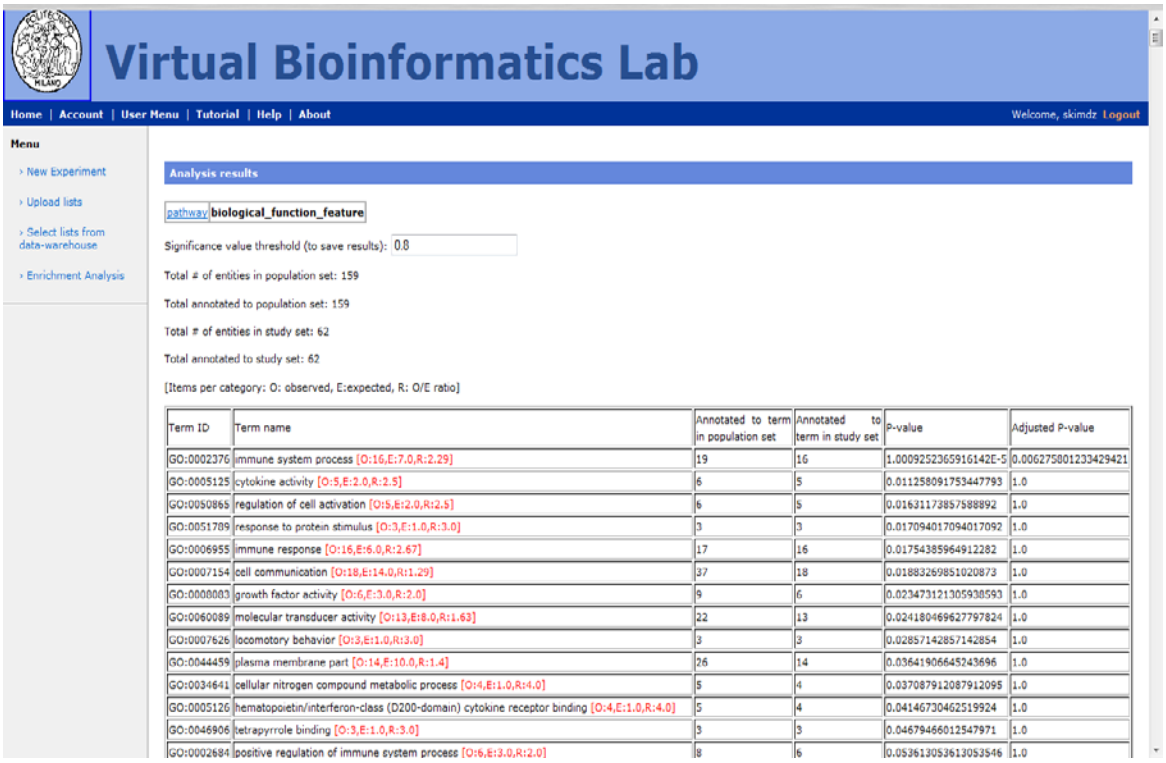


Figura 5.13. Schermata di scelta dei test statistci e delle correzioni di test multpli

In tale pagina è possibile selezionare uno dei test messi a disposizione per il calcolo della significatività per ogni feature da analizzare. Se il metodo scelto supporta la correzione per test multipli comparirà un menu di selezione. Per il metodo MGSA, per il quale non sono previste correzioni, invece, comparirà l'elenco dei parametri da inserire necessari per l'analisi (vengono mostrati sempre dei valori di default). Anche per quanto riguarda le correzioni, se necessario, sarà possibile inserire dei parametri. Una volta terminata la selezione delle impostazioni è possibile procedere con l'analisi.

6.5.2.3 Risultati dell'analisi

I risultati vengono mostrati in una tabella contenente ID, nome del termine, valori di significatività e conteggio dell'entità biomolecolari di population set e study set annotate a ogni termine (come visibile in fig. 6.14). Nella pagina sarà presente anche un elenco di tutte le feature biomediche analizzate: selezionandone una dall'elenco è possibile visualizzare la tabella con i risultati associati a essa.



Term ID	Term name	Annotated to term in population set	Annotated to term in study set	P-value	Adjusted P-value
GO:0002376	immune system process [O:16,E:7.0,R:2.29]	19	16	1.0009252365916142E-5	0.006275801233429421
GO:0005125	cytokine activity [O:5,E:2.0,R:2.5]	6	5	0.011258091753447793	1.0
GO:0050865	regulation of cell activation [O:5,E:2.0,R:2.5]	6	5	0.016311738575888892	1.0
GO:0051789	response to protein stimulus [O:3,E:1.0,R:3.0]	3	3	0.017094017094017092	1.0
GO:0006955	immune response [O:16,E:6.0,R:2.67]	17	16	0.01754385964912282	1.0
GO:0007154	cell communication [O:18,E:14.0,R:1.29]	37	18	0.01883269851020873	1.0
GO:0008083	growth factor activity [O:6,E:3.0,R:2.0]	9	6	0.023473121305938593	1.0
GO:0060089	molecular transducer activity [O:13,E:8.0,R:1.63]	22	13	0.024180469627797824	1.0
GO:0007626	locomotory behavior [O:3,E:1.0,R:3.0]	3	3	0.028571428571428554	1.0
GO:0044459	plasma membrane part [O:14,E:10.0,R:1.4]	26	14	0.03641906645243696	1.0
GO:0034641	cellular nitrogen compound metabolic process [O:4,E:1.0,R:4.0]	5	4	0.037087912087912095	1.0
GO:0005126	hematopoietin/interferon-class (D200-domain) cytokine receptor binding [O:4,E:1.0,R:4.0]	5	4	0.04146730462519924	1.0
GO:0046906	tetrapyrrole binding [O:3,E:1.0,R:3.0]	3	3	0.046794660125479 71	1.0
GO:0002684	positive regulation of immune system process [O:6,E:3.0,R:2.0]	8	6	0.053613053613053546	1.0

Figura 6.14 Schermata dei risultati

6.5.2.4 Gestione dei dati salvati

Per utenti avanzati, è possibile visualizzare i dati relativi a liste, esperimenti, analisi e risultati salvati. È possibile anche riutilizzare alcuni di questi dati: infatti si possono definire nuove analisi o nuovi esperimenti su liste caricate in precedenza, oppure rieseguire la stessa analisi su una versione più aggiornata della base di conoscenza. È possibile ovviamente rieseguire un'analisi variando alcuni parametri e inoltre confrontare risultati ottenuti con analisi diverse per una stessa feature biomedica. In figura 6.15 è mostrato un esempio di pagina web contenente la lista delle feature biomediche analizzate in un'analisi.

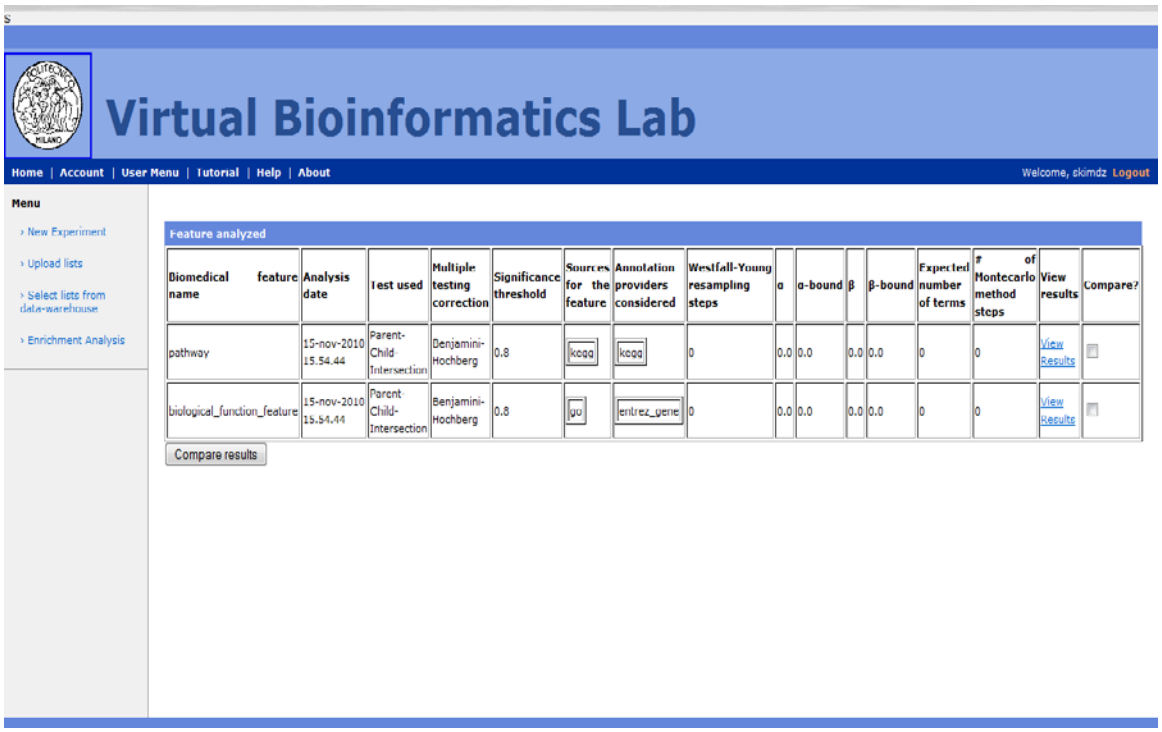


Figura 6.15. Pagina delle feature biomediche analizzate in una data analisi

Capitolo 7

Valutazione dei risultati

Per valutare i risultati ottenuti sono state effettuate delle analisi di test utilizzando alcune liste di geni di diverse dimensioni. Una prima lista utilizzata è composta da 107 ID di geni correlati alla patologia Diabete Mellito di cui 41 identificati come coinvolti nel Diabete Mellito dipendente dall'insulina e 66 coinvolti in Diabete Mellito non dipendente dall'insulina[48]. Una seconda lista è composta da 33 ID di geni correlati al morbo di Parkinson e a quello di Alzheimer[49] (18 geni espressi nel Parkinson e 15 nell'Alzheimer). Poiché queste liste contengono un numero esiguo di ID, è stato effettuato un test con una lista contenente 10000 ID di geni umani della sorgente Entrez Gene [50] estratti casualmente dalla base di conoscenza per valutare meglio le prestazioni in termini di tempo richiesto all'analisi (precisamente, come population set è stata presa l'intera lista, mentre come study set un sottoinsieme di 2000 elementi di tale lista). I test sono stati effettuati scegliendo di analizzare le annotazioni sui tre vocabolari controllati della Gene Ontology. Per valutare la bontà dei risultati ottenuti nell'analisi e delle prestazioni del sistema sono stati effettuati dei test con altri tool quali DAVID, Ontologizer e GFINDer utilizzando le stesse liste e sono state confrontate prestazioni (dove possibile) e qualità dei risultati delle analisi. Va premesso che ognuno di questi tool effettua le analisi in modi differenti e che le prestazioni velocistiche dipendono anche dalla potenza di calcolo a disposizione dalla macchina su cui è installato.

7.1 Prestazioni del sistema realizzato

Il sistema sviluppato, denominato GPEAer (Genomic and Proteomic Enrichment Analyzer), gira su un web server Tomcat installato su una macchina così composta: processore Intel Core 2 Duo P8400 a 2,26 GHz, 4 GB di RAM DDR3 a 1066 MHz, unità disco SATA a 5400 rpm e sistema operativo Windows 7 Professional a 64 Bit. Per quanto riguarda i test degli altri tool, Ontologizer è scaricabile dal Web ma gira in

locale, quindi sfrutta la macchina appena descritta; per quanto riguarda DAVID e GFINDer, essendo web application, tutto il carico di elaborazione è gestito dai loro server. Nel sistema sviluppato i tempi di risposta misurati sono relativi a due operazioni: la traduzione degli ID nel formato del GPDW e l'analisi di arricchimento di annotazioni.

Sistema	**Lunghezza liste**	**Tipi di analisi**					
		Non concettuale			**Concettuale**		
		Fisher Test	**Parent-child Intersection**	**MGSA**	**Fisher Test**	**Parent-child Intersection**	**MGSA**
GPEAer	107 ID	0:05	0:05	0:07	0:08	0:08	0:09
	66 ID	0:04	0:04	0:05	0:07	0:07	0:08
	10000 ID	0:05	0:05	0:06	0:08	0:08	0:09
DAVID	107 ID	n.d.	n.d.	n.d.	0:04 (*)	n.d.	n.d.
	66 ID	n.d.	n.d.	n.d.	0:04 (*)	n.d.	n.d.
	10000 ID	n.d.	n.d.	n.d.	0:08 (*)	n.d.	n.d.
Ontologizer	107 ID	0:06	0:06	0:07	n.d.	n.d.	n.d.
	66 ID	0:06	0:07	0:06	n.d.	n.d.	n.d.
	10000 ID	-	-	-	n.d.	n.d.	n.d.
GFINDer	107 ID	1:37	n.d.	n.d.	n.d.	n.d.	n.d.
	66 ID	1:29	n.d.	n.d.	n.d.	n.d.	n.d.
	10000 ID	-	n.d.	n.d.	n.d.	n.d.	n.d.

Tabella 7.3. Comparazione dei tempi di esecuzione di analisi di arricchimento dei tool considerati.
(*) DAVID esegue un'analisi di tipo concettuale solo lato entità biomolecolare.
n.d.: dati non disponibili perché la funzionalità non è implementata.
- : analisi non effettuata.
Tutti i tempi sono espressi in formato mm:ss.

In tabella 7.1 sono riportati i tempi medi (calcolati su 10 prove per ogni configurazione di test) per l'esecuzione dell'analisi di arricchimento; il processo di traduzione degli ID non è effettuata da Ontologizer e GFINDer, mentre per DAVID non è rilevabile poiché non è dichiarato il momento in cui effettivamente gli ID vengono convertiti nel formato del tool. Per i tool per cui era possibile, sono stati effettuati dei test utilizzando i vari algoritmi disponibili; come correzione di test multipli è stata usata la FDR Benjamini-Hochberg in tutte le analisi. Anche i tempi di esecuzione di Ontologizer sono riportati in tabella 7.1: è presente solo il tempo necessario all'analisi di arricchimento poiché Ontologizer non prevede la traduzione degli ID (peraltro accetta come input liste di simboli di geni e non di ID). Il tempo necessario alla traduzione degli ID nel sistema sviluppato è risultato di 2 sec. per le liste da 107 e 66 ID, di 3 sec. per la lista da 10000 ID. È probabile quindi che la maggior parte di questo tempo sia dovuta alla comunicazione con il web server, data la minima differenza di tempi per liste di dimensioni così diverse.

La prima prova è stata effettuata sulla lista di geni associati alla patologia Diabete Mellito; i tempi necessari a traduzione e analisi sono visibili in tabella 7.1. Le analisi di tipo non concettuale impiegano in media dai 5 ai 7 sec., in base al tipo di test effettuato; le analisi di tipo concettuale sia lato feature biomedica sia lato entità biomolecolare impiegano dagli 8 ai 9 sec. Come si può notare dai tempi, le analisi di tipo concettuale rallentano l'esecuzione, ma solo in minima parte. L'analisi della lista contenente gli ID di geni associati a Parkinson e Alzheimer ha dato risultati analoghi (dai 4 ai 5 sec. per analisi non concettuale, dai 7 agli 8 sec. per analisi concettuale). Si può notare che i tempi richiesti da Ontologizer sono molto simili a quelli del sistema sviluppato, dato che il pacchetto di analisi statistica utilizzato è pressoché identico; si nota invece che DAVID è leggermente più veloce. Per eseguire l'analisi sulle stesse liste con GFINDer utilizzando il Test di Fisher il tempo richiesto sale oltre il minuto, quindi notevolmente più elevato degli altri tre sistemi: ciò è giustificato dal fatto che tale tool risale al 2004 e non viene più aggiornato.

La lista da 10000 ID permette di ottenere ulteriori informazioni sulle prestazioni dei sistemi: non è stato effettuato il test su GFINDer, dati i già elevati tempi riscontrati per liste molto brevi; non è stato possibile effettuare il test su Ontologizer poiché tale tool accetta in input delle liste di simboli di geni (e non di ID) e la maggior parte dei simboli corrispondenti alla lista considerata non sono stati riconosciuti. Per effettuare le analisi di tale lista DAVID impiega un tempo piuttosto ridotto, solo 8 sec. Ciò è determinato in parte dalla potenza dei server su cui è installato il tool (CPU a 4 GHz e 8 GB di RAM) e in parte dal metodo utilizzato: DAVID, infatti, non effettua query a un database per l'estrazioni delle annotazioni, ma sfrutta degli oggetti Java molto grandi (di dimensioni fino a 2,5 GB) mantenuti in memoria. Le prestazioni del sistema sviluppato, sebbene il sistema non usi oggetti mantenuti in memoria, e la potenza della macchina è inferiore, risultano allineate: per analisi non concettuali si va dai 5 ai 6 secondi, per analisi concettuali dagli 8 ai 9 sec. Bisogna rilevare che il sistema realizzato memorizza OID di tipo concettuale durante la traduzione ma per estrarre le annotazioni è necessario convertirli in OID non concettuali del formato del data warehouse. Questa fase preliminare all'annotazione rallenta i tempi per le analisi di tipo concettuale, anche se in minima parte. Nei prossimi mesi sarà disponibile un livello di integrazione concettuale nella base di conoscenza e ciò permetterà di ridurre il numero di passaggi necessari per l'estrazione delle annotazioni, uniformando i tempi a quelli richiesti da DAVID. Si può quindi concludere che il sistema permette di effettuare analisi di arricchimento di tipo

concettuale e non concettuale per liste di ID biomolecolari numerosi garantendo prestazioni in linea con quelle offerte dai migliori tool esistenti.

7.2 Usabilità del sistema

Nello sviluppo del sistema, si è cercato di raggiungere una semplicità d'uso che permettesse l'utilizzo a tutti gli utenti, non necessariamente dotati di una preparazione informatica approfondita. L'applicazione web realizzata segue questo criterio e propone un percorso guidato per l'esecuzione di analisi di arricchimento. Ogni passo è definibile separatamente in una pagina web nell'ordine richiesto dall'analisi, e l'utente non esperto può quindi effettuare un'analisi con successo semplicemente completando i campi richiesti nelle pagine web. Per facilitare la selezione delle impostazioni da parte dell'utente, sono stati implementati dei controlli dinamici che permettono di visualizzare le possibilità di scelta in tempo reale. L'interfaccia web realizzata permette quindi di definire le impostazioni con una granularità molto fine e allo stesso tempo è molto intuitiva: sono sempre presenti dei parametri di default, in modo che gli utenti meno esperti possano effettuare tutte le operazioni senza perdere tempo nella configurazione più approfondita dell'analisi di arricchimento.

7.3 Risultati delle analisi di arricchimento implementate

Per dimostrare l'efficacia delle analisi di arricchimento realizzate in questa Tesi sono stati confrontati i risultati ottenuti con quelli degli altri tool considerati nei paragrafi precedenti. È da rilevare che i dati dei vari tool confrontati non sono aggiornati alla stessa versione; le liste analizzate però contengono geni ben noti da tempo, e la diversa versione dei dati non dovrebbe quindi influire in maniera significativa sugli esiti. In fig. 7.1 sono mostrati i risultati ottenuti da un'analisi effettuata sulla lista di ID di geni correlati alla patologia del Diabete Mellito utilizzando semplicemente il Test di Fisher, senza alcuna correzione per ontologie; come population set è stata selezionata l'intera lista, come study set il sottoinsieme di ID corrispondenti ai geni correlati al Diabete Mellito dipendente dall'insulina. In fig. 7.2 sono mostrati i risultati di un'analisi concettuale (sia lato entità biomolecolare sia lato feature biomedica) effettuata sempre

utilizzando il Test di Fisher senza correzioni per ontologie. Questi risultati sono stati confrontati con quelli provenienti da analisi effettuate con DAVID (che usa il Test di Fisher a una coda senza correzioni per ontologie, fig. 7.3) e con GFINDer (anch'esso usa il Test di Fisher a due code senza correzioni, fig. 7.4). Un ultimo confronto è stato fatto con Ontologizer, utilizzando sempre il Test di Fisher standard (risultati in fig. 7.5).

Analysis results

biological_function_feature

Significance value threshold (to save results): 0.8

Total # of entities in population set: 159

Total annotated to population set: 159

Total # of entities in study set: 62

Total annotated to study set: 62

[Items per category: O: observed, E:expected, R: O/E ratio]

Term ID	Term name	Annotated to term in population set	Annotated to term in study set	P-value	Adjusted P-value
GO:0006955	immune response [O:16,E:6.0,R:2.67]	17	16	9.107656443771486E-7	0.001053755850544361
GO:0002376	immune system process [O:16,E:7.0,R:2.29]	19	16	2.642876773106456E-5	0.015289042132420848
GO:0002250	adaptive immune response [O:5,E:1.0,R:5.0]	5	5	0.008142216010366719	0.7158193092452327
GO:0002460	adaptive immune response based on somatic recombination of immune receptors built from immunoglobulin superfamily domains [O:5,E:1.0,R:5.0]	5	5	0.008142216010366719	0.7158193092452327
GO:0006952	defense response [O:9,E:4.0,R:2.25]	12	9	0.011498193027975131	0.7158193092452327
GO:0050776	regulation of immune response [O:7,E:3.0,R:2.33]	9	7	0.021011398308153087	0.7158193092452327
GO:0009310	amine catabolic process [O:4,E:1.0,R:4.0]	4	4	0.021759370372533362	0.7158193092452327
GO:0009064	glutamine family amino acid metabolic process [O:4,E:1.0,R:4.0]	4	4	0.021759370372533362	0.7158193092452327
GO:0009063	cellular amino acid catabolic process [O:4,E:1.0,R:4.0]	4	4	0.021759370372533362	0.7158193092452327
GO:0009065	glutamine family amino acid catabolic process [O:4,E:1.0,R:4.0]	4	4	0.021759370372533362	0.7158193092452327
GO:0002703	regulation of leukocyte mediated immunity [O:4,E:1.0,R:4.0]	4	4	0.021759370372533362	0.7158193092452327
GO:0002706	regulation of lymphocyte mediated immunity [O:4,E:1.0,R:4.0]	4	4	0.021759370372533362	0.7158193092452327
GO:0050865	regulation of cell activation [O:5,E:2.0,R:2.5]	6	5	0.036798586644255865	0.7158193092452327

Figura 7.6. Risultati dell'analisi di tipo non concettuale effettuata dal sistema realizzato con Test di Fisher e correzione Benjamini-Hochberg sulla lista di ID di geni associati alla patologia Diabete Mellito

Analysis results

biological_function_feature

Significance value threshold (to save results): 0.8

Total # of entities in population set: 107

Total annotated to population set: 107

Total # of entities in study set: 41

Total annotated to study set: 41

[Items per category: O: observed, E:expected, R: O/E ratio]

Term ID	Term name	Annotated to term in population set	Annotated to term in study set	P-value	Adjusted P-value
GO:0006955	immune response [O:16,E:6.0,R:2.67]	17	16	3.061765615194032E-7	3.600636363468182E-4
GO:0002376	immune system process [O:16,E:7.0,R:2.29]	19	16	9.923330259449362E-6	0.005834918192556225
GO:0002250	adaptive immune response [O:5,E:1.0,R:5.0]	5	5	0.007049272021973964	0.7107520858515805
GO:0002460	adaptive immune response based on somatic recombination of immune receptors built from immunoglobulin superfamily domains [O:5,E:1.0,R:5.0]	5	5	0.007049272021973964	0.7107520858515805
GO:0006952	defense response [O:9,E:4.0,R:2.25]	12	9	0.008582475067007224	0.7107520858515805
GO:0050776	regulation of immune response [O:7,E:3.0,R:2.33]	9	7	0.017162945113954272	0.7107520858515805
GO:0004871	signal transducer activity [O:17,E:11.0,R:1.55]	30	17	0.018059796682982777	0.7107520858515805
GO:0060089	molecular transducer activity [O:17,E:11.0,R:1.55]	30	17	0.018059796682982777	0.7107520858515805
GO:0009063	cellular amino acid catabolic process [O:4,E:1.0,R:4.0]	4	4	0.019623649142251613	0.7107520858515805
GO:0009064	glutamine family amino acid metabolic process [O:4,E:1.0,R:4.0]	4	4	0.019623649142251613	0.7107520858515805
GO:0009065	glutamine family amino acid catabolic process [O:4,E:1.0,R:4.0]	4	4	0.019623649142251613	0.7107520858515805
GO:0002703	regulation of leukocyte mediated immunity [O:4,E:1.0,R:4.0]	4	4	0.019623649142251613	0.7107520858515805
GO:0002706	regulation of lymphocyte mediated immunity [O:4,E:1.0,R:4.0]	4	4	0.019623649142251613	0.7107520858515805

Figura 7.7.Risultati dell'analisi concettuale effettuata dal sistema realizzato con Test di Fisher e correzione Benjamini-Hochberg sulla lista di ID di geni associati alla patologia Diabete Mellito

1346 chart records — Download File — 1, 2

Sublist	Category	Term	RT	Genes	Count	%	P-Value	Benjamini
☐	GOTERM_BP_ALL	immune response	RT		15	37,5	1,1E-4	1,1E-1
☐	GOTERM_BP_ALL	immune system process	RT		15	37,5	1,8E-3	6,0E-1
☐	GOTERM_CC_ALL	plasma membrane part	RT		17	42,5	2,7E-2	9,7E-1
☐	GOTERM_BP_ALL	response to biotic stimulus	RT		7	17,5	5,8E-2	1,0E0
☐	GOTERM_MF_ALL	cytokine activity	RT		6	15,0	6,1E-2	1,0E0
☐	GOTERM_MF_ALL	signal transducer activity	RT		13	32,5	6,7E-2	1,0E0
☐	GOTERM_MF_ALL	molecular transducer activity	RT		13	32,5	6,7E-2	1,0E0
☐	GOTERM_MF_ALL	receptor activity	RT		12	30,0	7,9E-2	1,0E0
☐	GOTERM_BP_ALL	locomotion	RT		6	15,0	1,2E-1	1,0E0
☐	GOTERM_BP_ALL	cell activation	RT		6	15,0	1,2E-1	1,0E0
☐	GOTERM_BP_ALL	adaptive immune response based on somatic recombination of immune receptors built from immunoglobulin superfamily domains	RT		5	12,5	1,3E-1	1,0E0
☐	GOTERM_BP_ALL	adaptive immune response	RT		5	12,5	1,3E-1	1,0E0
☐	GOTERM_BP_ALL	regulation of cell activation	RT		5	12,5	1,3E-1	1,0E0
☐	GOTERM_BP_ALL	glutamine family amino acid catabolic process	RT		4	10,0	1,4E-1	1,0E0
☐	GOTERM_BP_ALL	amine catabolic process	RT		4	10,0	1,4E-1	1,0E0
☐	GOTERM_BP_ALL	glutamine family amino acid metabolic process	RT		4	10,0	1,4E-1	1,0E0
☐	GOTERM_BP_ALL	cellular amino acid catabolic process	RT		4	10,0	1,4E-1	1,0E0

Figura 7.8.Risultati dell'analisi effettuata con Test di Fisher e correzione Benjamini-Hochberg sulla lista di ID di geni associati alla patologia Diabete Mellito da DAVID

GFINDer: Genome Function INtegrated Discoverer

GO Path Code	GO Level	GO Category name	P-value$_{test\text{-}type}$	Log(1/P)
1.20.4 / 1.8.6	3	immune response [O:15, E:6.27, R:2.39] biological_process	P_F=0.00001	
1.8	2	immune system process [O:15, E:7.75, R:1.94] biological_process	P_F=0.00036	
1.20.17.2	4	defense response [O:11, E:5.17, R:2.13] biological_process	P_F=0.00085	
1.20.4.4 / 1.8.6.4 / 1.8.5	3 4	immune effector process [O:7, E:2.95, R:2.37] biological_process	P_F=0.00366	
1.20.4.1 / 1.8.6.1	4	adaptive immune response [O:5, E:1.84, R:2.72] biological_process	P_F=0.00573	
1.20.4.1.1 / 1.8.6.1.1	5	adaptive immune response based on somatic recombination of immune receptors built from immunoglobulin superfamily domains [O:5, E:1.84, R:2.72] biological_process	P_F=0.00573	
1.8.5.8 / 1.8.6.4.8 / 1.20.4.4.8	4 5	leukocyte mediated immunity [O:5, E:1.84, R:2.72] biological_process	P_F=0.00573	
1.8.5.8.1 / 1.8.6.4.8.1 / 1.20.4.4.8.1	5 6	lymphocyte mediated immunity [O:5, E:1.84, R:2.72] biological_process	P_F=0.00573	
1.17.6.6 / 1.2.1.6.6 / 1.4.3.12 / 1.4.45.6	4 5	regulation of cell activation [O:5, E:1.84, R:2.72] biological_process	P_F=0.00573	

Figura 7.9.Risultati dell'analisi effettuata con Test di Fisher e correzione FDR sulla lista di ID di geni associati alla patologia Diabete Mellito da GFINDer

Display terms emanating from Gene Ontology

Table

GO ID	Name	NSP	P-Value	Adj. P-Value	Rank	Pop. Count	Study Count
GO:0006955	immune response	B	0.000278	0.518	1	20	15
GO:0002376	immune system process	B	0.00150	0.704	2	24	16
GO:0009607	response to biotic stimulus	B	0.00677	0.704	3	10	8
GO:0002699	positive regulation of immune effector pro...	B	0.00727	0.704	4	5	5
GO:0045087	innate immune response	B	0.00727	0.704	5	5	5
GO:0051707	response to other organism	B	0.0127	0.704	6	7	6
GO:0002443	leukocyte mediated immunity	B	0.0127	0.704	7	7	6
GO:0002449	lymphocyte mediated immunity	B	0.0127	0.704	8	7	6
GO:0005125	cytokine activity	M	0.0127	0.704	9	7	6
GO:0001775	cell activation	B	0.0159	0.704	10	9	7
GO:0006952	defense response	B	0.0167	0.704	11	17	11
GO:0002252	immune effector process	B	0.0174	0.704	12	11	8
GO:0080134	regulation of response to stress	B	0.0174	0.704	13	11	8
GO:0009063	cellular amino acid catabolic process	B	0.0201	0.704	14	4	4
GO:0009064	glutamine family amino acid metabolic pr...	B	0.0201	0.704	15	4	4

0/0/1862

Figura 7.10.Risultati dell'analisi effettuata con Test di Fisher e correzione Benjamini-Hochberg sulla lista di ID di geni associati alla patologia Diabete Mellito da Ontologizer

Si può notare che le analisi di tipo concettuale e non concettuale presentano i primi 6 termini comuni, e condividono quasi tutti gli altri; i valori invece cambiano, e precisamente i primi termini presentano un p-value inferiore nell'analisi concettuale (quindi risultano più significativi nel set di geni considerato). Confrontando con i risultati ottenuti da DAVID, si notano invece alcune differenze: i due termini più significativi sono ancora gli stessi, molti altri termini sono comuni, ma sono anche presenti dei termini diversi e un ordine di rilevanza diverso. Ciò è probabilmente dovuto ai differenti algoritmi implementati. Nel confronto con GFINDer e Ontologizer i termini più significativi risultano nelle stesse posizioni. I termini più significativi in questo studio sono quindi risultati *immune response* (GO:0006955) e *immune system process* (GO:0002376) con ogni tool considerato. Sono state quindi effettuate delle analisi sulla stessa lista utilizzando alcuni degli altri algoritmi disponibili nel sistema sviluppato: i risultati dell'analisi con l'algoritmo Parent-Child Intersection (correzione per ontologie di Grossmann) sono illustrati in fig. 7.6; i risultati dell'analisi di tipo MGSA (effettuata utilizzando i parametri di default) sono illustrati in fig. 7.7. In entrambi i casi è stata effettuata un'analisi di tipo concettuale.

Analysis results

biological_function_feature

Significance value threshold (to save results): 0.8

Total # of entities in population set: 107

Total annotated to population set: 107

Total # of entities in study set: 41

Total annotated to study set: 41

[Items per category: O: observed, E:expected, R: O/E ratio]

Term ID	Term name	Annotated to term in population set	Annotated to term in study set	P-value	Adjusted P-value
GO:0002376	immune system process [O:16,E:7.0,R:2.29]	19	16	7.210134038887763E-6	0.00454238444449291
GO:0005125	cytokine activity [O:5,E:2.0,R:2.5]	6	5	0.012332015810276643	1.0
GO:0060089	molecular transducer activity [O:17,E:11.0,R:1.55]	30	17	0.014594485998579501	1.0
GO:0050865	regulation of cell activation [O:5,E:2.0,R:2.5]	6	5	0.01631173857588892	1.0
GO:0051789	response to protein stimulus [O:3,E:1.0,R:3.0]	3	3	0.017094017094017092	1.0
GO:0006955	immune response [O:16,E:6.0,R:2.67]	17	16	0.01754385964912282	1.0
GO:0044459	plasma membrane part [O:14,E:9.0,R:1.56]	26	14	0.020120560443141068	1.0
GO:0008083	growth factor activity [O:6,E:3.0,R:2.0]	9	6	0.0273963215408901 2	1.0
GO:0007626	locomotory behavior [O:3,E:1.0,R:3.0]	3	3	0.02857142857142854	1.0
GO:0034641	cellular nitrogen compound metabolic process [O:4,E:1.0,R:4.0]	5	4	0.037087912087912095	1.0
GO:0046908	negative regulation of crystal formation [O:4,E:3.0,R:1.33]	9	4	0.04117647058823562	1.0
GO:0050801	ion homeostasis [O:4,E:3.0,R:1.33]	9	4	0.04117647058823562	1.0
GO:0005126	hematopoietin/interferon-class (D200-domain) cytokine receptor binding [O:4,E:1.0,R:4.0]	5	4	0.042687747035572765	1.0
GO:0046906	tetrapyrrole binding [O:3,E:1.0,R:3.0]	3	3	0.04842647856754966	1.0

Figura 7.11.Risultati dell'analisi concettuale effettuata con algoritmo Parent-Child Intersection e correzione Benjamini-Hochberg sulla lista di ID di geni associati alla patologia Diabete Mellito dal sistema sviluppato

Analysis results

biological_function_feature

Significance value threshold (to save results): 0.8

Total # of entities in population set: 107

Total annotated to population set: 107

Total # of entities in study set: 41

Total annotated to study set: 41

[Items per category: O: observed, E:expected, R: O/E ratio]

Term ID	Term name	Annotated to term in population set	Annotated to term in study set	Marginal
GO:0008289	lipid binding [O:4,E:5.0,R:0.8]	14	4	1.0
GO:0044262	cellular carbohydrate metabolic process [O:4,E:4.0,R:1.0]	13	4	1.0
GO:0042303	molting cycle [O:1,E:1.0,R:1.0]	4	1	1.0
GO:0044248	cellular catabolic process [O:5,E:4.0,R:1.25]	12	5	0.9484144758634914
GO:0009653	anatomical structure morphogenesis [O:2,E:3.0,R:0.67]	10	2	0.8691018106287721
GO:0030054	cell junction [O:2,E:1.0,R:2.0]	4	2	0.6890264354717406
GO:0003700	transcription factor activity [O:2,E:4.0,R:0.5]	11	2	0.619999208331684
GO:0022414	reproductive process [O:3,E:3.0,R:1.0]	9	3	0.4586842889256019
GO:0051234	establishment of localization [O:4,E:8.0,R:0.5]	23	4	0.4337571536607368
GO:0000003	reproduction [O:3,E:3.0,R:1.0]	9	3	0.4236321325669428
GO:0030528	transcription regulator activity [O:2,E:4.0,R:0.5]	11	2	0.38061954295738115
GO:0016053	organic acid biosynthetic process [O:1,E:1.0,R:1.0]	4	1	0.3654757614078363
GO:0050796	regulation of insulin secretion [O:1,E:1.0,R:1.0]	4	1	0.3409944604051258
GO:0008610	lipid biosynthetic process [O:1,E:3.0,R:0.33]	8	1	0.33154027404223757

Figura 7.12.Risultati dell'analisi concettuale effettuata con algoritmo MGSA sulla lista di ID di geni associati alla patologia Diabete Mellito dal sistema sviluppato

Si può notare che l'analisi con correzione di Grossmann restituisce dei risultati leggermente diversi dai precedenti: il termine *immune response* è infatti in una posizione inferiore, mentre i termini *cytokine activity* e *molecular transducer activity*

risultano più significativi (similmente ai risultati ottenuti con DAVID). I risultati dall'analisi di tipo MGSA sono invece molto diversi; MGSA, infatti, come riportato in [42], fornisce dei risultati di alto livello relativi al nucleo dei processi biologici, eliminando risultati ridondanti.

Sono stati confrontati i risultati ottenuti con i vari tool considerati anche per l'esperimento basato sulla lista contenente ID di geni correlati a morbo di Parkinson e Alzheimer. Come population set è stata utilizzata l'intera lista, come study set l'insieme degli ID di geni correlati all'Alzheimer. In questo caso i risultati confrontati a quelli degli altri tool non corrispondono perfettamente per quanto riguarda le primissime categorie più significative, ma comunque la maggior parte dei termini più rilevanti sono presenti nell'elenco dei risultati. Alcuni risultati derivanti dalle analisi effettuate dal sistema sviluppato, da DAVID e da GFINDER sono illustrati rispettivamente nelle figure 7.8, 7.9, 7.10 e 7.11.

Analysis results

biological_function_feature

Significance value threshold (to save results): 0.8

Total # of entities in population set: 48

Total annotated to population set: 48

Total # of entities in study set: 20

Total annotated to study set: 20

[Items per category: O: observed, E:expected, R: O/E ratio]

Term ID	Term name	Annotated to term in population set	Annotated to term in study set	P-value	Adjusted P-value
GO:0044459	plasma membrane part [O:6,E:2.0,R:3.0]	6	6	0.0031585349873755534	0.5901486988847592
GO:0065008	regulation of biological quality [O:7,E:3.0,R:2.33]	8	7	0.0064197865597064915	0.5901486988847592
GO:0007166	cell surface receptor linked signal transduction [O:5,E:2.0,R:2.5]	5	5	0.0090544669638099 77	0.5901486988847592
GO:0005576	extracellular region [O:6,E:2.0,R:3.0]	7	6	0.01684551993266985	0.5901486988847592
GO:0048519	negative regulation of biological process [O:7,E:3.0,R:2.33]	9	7	0.021878632595479935	0.5901486988847592
GO:0009056	catabolic process [O:8,E:4.0,R:2.0]	11	8	0.024356399900724236	0.5901486988847592
GO:0032101	regulation of response to external stimulus [O:4,E:1.0,R:4.0]	4	4	0.024899784150477566	0.5901486988847592
GO:0009986	cell surface [O:4,E:1.0,R:4.0]	4	4	0.024899784150477566	0.5901486988847592
GO:0044421	extracellular region part [O:4,E:1.0,R:4.0]	4	4	0.024899784150477566	0.5901486988847592
GO:0080134	regulation of response to stress [O:4,E:1.0,R:4.0]	4	4	0.024899784150477566	0.5901486988847592
GO:0044093	positive regulation of molecular function [O:4,E:1.0,R:4.0]	4	4	0.024899784150477566	0.5901486988847592
GO:0006916	anti-apoptosis [O:4,E:1.0,R:4.0]	4	4	0.024899784150477566	0.5901486988847592
GO:0048583	regulation of response to stimulus [O:4,E:1.0,R:4.0]	4	4	0.024899784150477566	0.5901486988847592
GO:0051093	negative regulation of developmental process [O:4,E:1.0,R:4.0]	4	4	0.024899784150477566	0.5901486988847592

Figura 7.13. Risultati dell'analisi non concettuale effettuata con Test di Fisher e correzione Benjamini-Hochberg sulla lista di ID di geni associati alle patologie Parkinson e Alzheimer dal sistema sviluppato

Analysis results

biological_function_feature

Significance value threshold (to save results): 0.8

Total # of entities in population set: 32

Total annotated to population set: 32

Total # of entities in study set: 15

Total annotated to study set: 15

[Items per category: O: observed, E:expected, R: O/E ratio]

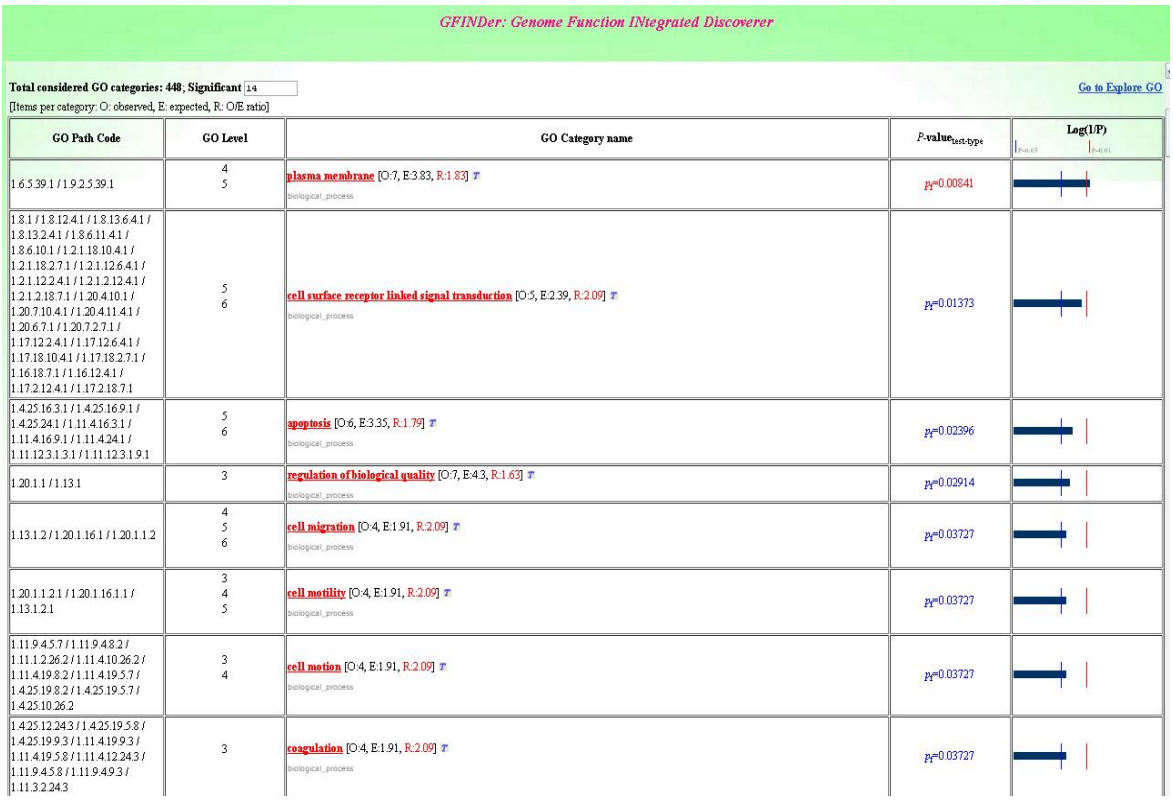
Term ID	Term name	Annotated to term in population set	Annotated to term in study set	P-value	Adjusted P-value
GO:0009056	catabolic process [O:8,E:5.0,R:1.6]	11	8	0.002183059458601253	0.7466063348416285
GO:0065008	regulation of biological quality [O:8,E:4.0,R:2.0]	9	8	0.0078157136980666681	1.0
GO:0044459	plasma membrane part [O:6,E:2.0,R:3.0]	6	6	0.011904761904761887	1.0
GO:0005576	extracellular region [O:6,E:3.0,R:2.0]	7	6	0.014828375286041053	1.0
GO:0004625	calcium-dependent secreted phospholipase A2 activity [O:6,E:3.0,R:2.0]	7	6	0.014828375286041053	1.0
GO:0009986	cell surface [O:4,E:1.0,R:4.0]	4	4	0.026086956521738997	1.0
GO:0009100	glycoprotein metabolic process [O:2,E:0.0,R:2.1474836E7]	2	2	0.035714285714285705	1.0
GO:0005886	plasma membrane [O:7,E:4.0,R:1.75]	10	7	0.03636328673345766	1.0
GO:0007166	cell surface receptor linked signal transduction [O:5,E:2.0,R:2.5]	5	5	0.04545454545454544	1.0
GO:0030246	carbohydrate binding [O:3,E:1.0,R:3.0]	3	3	0.04743083003952567	1.0
GO:0005102	receptor binding [O:3,E:1.0,R:3.0]	3	3	0.06315789473684225	1.0
GO:0048583	regulation of response to stimulus [O:4,E:1.0,R:4.0]	4	4	0.06862745098039238	1.0
GO:0065007	biological regulation [O:10,E:8.0,R:1.25]	18	10	0.08299771167048017	1.0
GO:0048519	negative regulation of biological process [O:7,E:4.0,R:1.75]	9	7	0.08432743726861426	1.0
GO:0012505	endomembrane system [O:4,E:2.0,R:2.0]	5	4	0.10434782608695617	1.0

Figura 7.14. Risultati dell'analisi concettuale effettuata con algoritmo Parent-Child Intersection e correzione Benjamini-Hochberg sulla lista di ID di geni associati alle patologie Parkinson e Alzheimer dal sistema sviluppato

502 chart records — Download File

Sublist	Category	Term	RT	Genes	Count	%	P-Value	Benjamini
☐	GOTERM_BP_ALL	regulation of response to stimulus	RT		6	40,0	3,5E-2	1,0E0
☐	GOTERM_BP_ALL	response to chemical stimulus	RT		9	60,0	5,7E-2	1,0E0
☐	GOTERM_BP_ALL	anatomical structure morphogenesis	RT		8	53,3	7,4E-2	1,0E0
☐	GOTERM_BP_ALL	negative regulation of biological process	RT		8	53,3	7,4E-2	1,0E0
☐	GOTERM_BP_ALL	organ development	RT		7	46,7	8,6E-2	1,0E0
☐	GOTERM_BP_ALL	positive regulation of apoptosis	RT		5	33,3	8,9E-2	1,0E0
☐	GOTERM_BP_ALL	cell surface receptor linked signal transduction	RT		5	33,3	8,9E-2	1,0E0
☐	GOTERM_BP_ALL	regulation of coagulation	RT		5	33,3	8,9E-2	1,0E0
☐	GOTERM_CC_ALL	extracellular region	RT		6	40,0	9,2E-2	1,0E0
☐	GOTERM_CC_ALL	plasma membrane part	RT		6	40,0	9,2E-2	1,0E0
☐	GOTERM_MF_ALL	peptidase activity	RT		6	40,0	9,2E-2	1,0E0
☐	GOTERM_MF_ALL	hydrolase activity	RT		6	40,0	9,2E-2	1,0E0
☐	GOTERM_BP_ALL	regulation of metabolic process	RT		8	53,3	1,4E-1	1,0E0
☐	GOTERM_BP_ALL	regulation of primary metabolic process	RT		8	53,3	1,4E-1	1,0E0
☐	GOTERM_BP_ALL	response to stimulus	RT		10	66,7	1,4E-1	1,0E0
☐	GOTERM_BP_ALL	negative regulation of cellular process	RT		7	46,7	1,6E-1	1,0E0

Figura 7.15. Risultati dell'analisi effettuata con Test di Fisher e correzione Benjamini-Hochberg sulla lista di ID di geni associati alle patologie Parkinson e Alzheimer da DAVID



GFINDer: Genome Function INtegrated Discoverer

Total considered GO categories: 448; Significant 14 — Go to Explore GO

[Items per category: O: observed, E: expected, R: O/E ratio]

GO Path Code	GO Level	GO Category name	P-value$_{test\text{-}type}$	Log(1/P)
1.6.5.39.1 / 1.9.2.5.39.1	4 5	**plasma membrane** [O:7, E:3.83, R:1.83] biological_process	p_F=0.00841	
1.8.1 / 1.8.12.4.1 / 1.8.13.6.4.1 / 1.8.13.2.4.1 / 1.8.6.11.4.1 / 1.8.6.10.1 / 1.2.1.18.10.4.1 / 1.2.1.18.2.7.1 / 1.2.1.12.6.4.1 / 1.2.1.12.2.4.1 / 1.2.1.2.12.4.1 / 1.2.1.2.18.7.1 / 1.20.4.10.1 / 1.20.7.10.4.1 / 1.20.4.11.4.1 / 1.20.6.7.1 / 1.20.7.2.7.1 / 1.17.12.2.4.1 / 1.17.12.6.4.1 / 1.17.18.10.4.1 / 1.17.18.2.7.1 / 1.16.18.7.1 / 1.16.12.4.1 / 1.17.2.12.4.1 / 1.17.2.18.7.1	5 6	**cell surface receptor linked signal transduction** [O:5, E:2.39, R:2.09] biological_process	p_F=0.01373	
1.4.25.16.3.1 / 1.4.25.16.9.1 / 1.4.25.24.1 / 1.11.4.16.3.1 / 1.11.4.16.9.1 / 1.11.4.24.1 / 1.11.12.3.1.3.1 / 1.11.12.3.1.9.1	5 6	**apoptosis** [O:6, E:3.35, R:1.79] biological_process	p_F=0.02396	
1.20.1.1 / 1.13.1	3	**regulation of biological quality** [O:7, E:4.3, R:1.63] biological_process	p_F=0.02914	
1.13.1.2 / 1.20.1.16.1 / 1.20.1.1.2	4 5 6	**cell migration** [O:4, E:1.91, R:2.09] biological_process	p_F=0.03727	
1.20.1.1.2.1 / 1.20.1.16.1.1 / 1.13.1.2.1	3 4 5	**cell motility** [O:4, E:1.91, R:2.09] biological_process	p_F=0.03727	
1.11.9.4.5.7 / 1.11.9.4.8.2 / 1.11.1.2.26.2 / 1.11.4.10.26.2 / 1.11.4.19.8.2 / 1.11.4.19.5.7 / 1.4.25.19.8.2 / 1.4.25.19.5.7 / 1.4.25.10.26.2	3 4	**cell motion** [O:4, E:1.91, R:2.09] biological_process	p_F=0.03727	
1.4.25.12.24.3 / 1.4.25.19.5.8 / 1.4.25.19.9.3 / 1.11.4.19.9.3 / 1.11.4.19.5.8 / 1.11.4.12.24.3 / 1.11.9.4.5.8 / 1.11.9.4.9.3 / 1.11.3.2.24.3	3	**coagulation** [O:4, E:1.91, R:2.09] biological_process	p_F=0.03727	

Figura 7.16.Risultati dell'analisi effettuata con Test di Fisher e correzione FDR sulla lista di ID di geni associati alle patologie Parkinson e Alzheimer da GFINDer

È importante sottolineare che i risultati delle analisi dipendono fortemente dal metodo con cui esse sono implementate: il sistema sviluppato e GFINDer, nel calcolo statistico, contano tutti gli ID presenti nelle liste, Ontologizer e DAVID invece contano solo gli ID che possiedono almeno un'annotazione.

Il sistema sviluppato in questa Tesi è risultato quindi uno strumento valido per le analisi di arricchimento di annotazioni biomolecolari, e offre una grande varietà di algoritmi di analisi, in modo da poter ottenere risultati più completi.

Capitolo 8

Conclusioni

In questa Tesi è stato affrontato il problema di progettare sistemi informativi web che possano gestire efficientemente grandi quantità di dati provenienti da una base di conoscenza e di dati generati dagli utenti e correlati a quelli di tale base di conoscenza. È stato inoltre affrontato il problema della conservazione di tale correlazione e del mantenimento dell'operatività del sistema informativo durante gli aggiornamenti della base di conoscenza. Questi aspetti sono stati studiati in un particolare caso, quello bioinformatico, che presenta criticità molto elevate nei requisiti: grandi moli di dati provenienti da una base di conoscenza in evoluzione, numerosi dati generati dagli utenti e aggiornamenti molto frequenti dei dati. In particolare, è stato considerato il caso di un sistema per analisi di arricchimento di annotazioni biomolecolari, che rispetti tali requisiti. È stato quindi progettato e realizzato un sistema web che, tramite servizi e un'applicazione web di facile utilizzo che usa tali servizi, fornisce un efficace strumento per l'analisi di arricchimento di annotazioni biomolecolari. L'utilizzo di una grande base di conoscenza quale la GPKB, contenente annotazioni genomiche e proteomiche integrate provenienti dalle maggiori banche dati mondiali, e aggiornata con una frequenza elevata, permette al sistema di effettuare analisi di arricchimento ottenendo risultati più significativi rispetto ad analisi effettuate da sistemi che non utilizzano basi di conoscenza di questo tipo. Il sistema progettato permette di gestire efficacemente i numerosi dati utilizzati e memorizzati in modo efficiente e integra un meccanismo per conservare la correlazione fra i dati degli utenti e quelli della base di conoscenza durante una sua evoluzione.

Per raggiungere questi obiettivi prefissati, sono stati ottenuti i seguenti risultati rilevanti:

1. sviluppo di un metodo per l'estrazione e la gestione efficiente dei dati provenienti da una base di conoscenza e dei numerosi dati generati dagli utenti che usano i servizi o le applicazioni basati su di essa

2. realizzazione di un algoritmo per l'aggiornamento dei dati degli utenti del sistema informativo sviluppato, in modo da mantenere la correlazione di tali dati con quelli presenti nella base di conoscenza e da ridurre al minimo i tempi di inutilizzabilità del sistema da parte di ogni singolo utente
3. realizzazione di servizi web che permettono di effettuare analisi di arricchimento di annotazioni biomolecolari in modo efficiente, sfruttando i dati presenti nella GPKB e le implementazioni degli algoritmi di analisi statistica offerte da Ontologizer
4. progettazione e implementazione di un database di back-end dei servizi che memorizza i numerosi dati degli utenti di tali servizi e i risultati derivanti dalle analisi in modo efficiente sia in termini di spazio occupato, sia in termini di tempo richiesto per l'interrogazione e l'estrazione di tali dati
5. progettazione e realizzazione di un'applicazione web dinamica e di facile utilizzo, anche per utenti con limitata preparazione informatica, che dimostra il funzionamento dei servizi web sviluppati.

Data la vastità dei dati biomedici e biomolecolari messi a disposizione dalla Genomic and Proteomic Knowledge Base e il loro costante aggiornamento, l'efficiente sistema bioinformatico realizzato in questa Tesi rappresenta un importante strumento per ricercatori biomedici a supporto della comprensione dei risultati di esperimenti biomolecolari e dei complessi processi biologici.

L'utilizzo di un'architettura orientata ai servizi consente inoltre il riutilizzo e l'integrazione dei servizi web sviluppati in processi automatici per effettuare anche altre analisi più complesse.

Il metodo studiato per la gestione di grandi moli di dati (sia della base di conoscenza, sia generati dagli utenti del sistema) e per la gestione dell'evoluzione di una base di conoscenza correlata a tali dati, che qui è stato applicato nel caso bioinformatico considerato, per le sue caratteristiche generali potrà inoltre essere utilizzato efficacemente anche negli altri ambiti informatici.

Capitolo 9

Sviluppi futuri

Il sistema sviluppato in questa Tesi è stato progettato per l'esecuzione di analisi di arricchimento da parte di singoli utenti. Ciascun utente registrato può caricare le proprie liste di ID e analizzarle, ma non può accedere a dati caricati o analizzati da altri utenti. Uno sviluppo futuro può consistere nel rendere il sistema collaborativo, consentendo agli utenti di condividere i propri dati, sia liste di identificativi biomolecolari, sia impostazioni di analisi e risultati ottenuti; in tal caso i dati condivisi potrebbero essere riutilizzati anche da altri utenti per definire nuovi esperimenti e realizzare nuove analisi.
Un'altra possibile estensione al lavoro svolto è la possibilità di selezionare liste di ID di entità biomolecolari direttamente dalla base di conoscenza, invece che caricarle manualmente, filtrando gli ID di entità biomolecolari presenti nella base di conoscenza tramite gli attributi a essi associati (ad esempio la sorgente di provenienza, la tassonomia, etc.), oppure selezionando un sottoinsieme di ID che siano annotati a una o più date feature biomediche. Tale estensione è già stata progettata durante questa Tesi, ma non è ancora stata implementata nel sistema.
Un ulteriore sviluppo possibile è rappresentato dall'ampliamento dei servizi disponibili: per ora è stato sviluppato un servizio di analisi di arricchimento di annotazioni biomolecolari e alcuni semplici servizi di look-up di dati e metadati della GPKB; in futuro potranno essere sviluppati ulteriori servizi, ad esempio:

- un servizio per l'esplorazione delle categorie di annotazione relative a liste di ID di entità biomolecolari per ogni ontologia e terminologia disponibile nella GPKB
- un servizio per il recupero e la visualizzazione di annotazioni di entità biomolecolari, disgiunto dall'analisi di arricchimento
- dei servizi per l'analisi di similarità funzionale di entità biomolecolari e per la predizione di annotazioni

- un servizio per la conversione di identificativi di entità biomolecolari provenienti da una sorgente dati specificata dall'utente in equivalenti identificativi di un'altra sorgente dati.

Si potrebbe inoltre sviluppare un sistema che integri tali e altri servizi per offrire la possibilità di realizzare analisi, anche automatiche, più complesse.

Un'ultima miglioria al sistema può riguardare l'applicazione web sviluppata: si potrebbe rendere dinamica e interattiva la visualizzazione dei risultati, anche integrandoli su grafici rappresentanti i termini delle ontologie analizzate nel caso l'analisi sia realizzata su annotazioni ontologiche.

Capitolo 10

Bibliografia

1. Affymetrix. [Online]. Disponibile su: URL: http://www.affymetrix.com/estore/
2. Cochrane GR, Galperin MY. The 2010 Nucleic Acids Research database issue and online database collection: a community of data resources. *Nucleic Acids Research* 2009 Dec 3; 38:D1-D4.
3. KEGG: Kyoto Encyclopedia of Genes and Genomes. [Online]. Disponibile su: URL: http://www.genome.jp/kegg/
4. InterPro protein sequence analysis & classification. [Online]. Disponibile su: URL: http://www.ebi.ac.uk/interpro/
5. Gold DL, Coombes KR, Wang J, Mallick B. Enrichment analysis in high-throughput genomics – accounting for dependency in the NULL. *Briefings in bioinformatics* 2006 Oct 31; 8 (2): 71-77.
6. Soliani L. Manuale di statistica per la ricerca e la professione: statistica univariata e bivariata parametrica e non-parametrica per le discipline ambientali e biologiche. [Online]. 2005; Disponibile su: URL: http://www.dsa.unipr.it/soliani/soliani.html
7. Rivals I, Personnaz L, Taing L, Potier MC. Enrichment or depletion of a GO category within a class of genes: which test? *Bioinformatics* 2006 Dec 20; 23(4):401-407.
8. Agresti A. A survey of exact inference for contingency tables. *Statistical Science* 1992; 7(1), 131–177.
9. Abdi H. Bonferroni and Sidak corrections for multiple comparisons. In: Salkind NJ, editor. Encyclopedia of Measurement and Statistics. Thousand Oaks (CA): Sage; 2007. pp. 103-107.
10. Abdi H. Holm's sequential Bonferroni procedure. In: Salkind NJ, Dougherty DM, Frey B, editors: Encyclopedia of Research Design. Thousand Oaks (CA): Sage; 2010. pp. 573-577.

11. Cox DD, Lee J. Pointwise testing with functional data using the Westfall-Young randomization method. *Biometrika* 2008; 95: 621-634.
12. Multiple testing correction. [Online]. Disponibile su: URL: http://www.silicongenetics.com/Support/GeneSpring/GSnotes/analysis_guides/mtc.pdf
13. Benjamini Y, Hochberg Y. Controlling the False Discovery Rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistics Society B* 1995;57:289-300.
14. DAVID. The Database for Annotation, Visualization and Integrated Discovery. [Online]. Disponibile su: URL: http://david.abcc.ncifcrf.gov/
15. Huang DW, Sherman BT, Tan Q, Kir J, Liu D, Bryant D, Guo Y, Stephens R, Baseler MW,Lane HC, Lempicki RA. DAVID Bioinformatics Resources: expanded annotation database and novel algorithms to better estract biology from large gene lists. *Nucleic Acids Research*. 2007;35:W 169-W175.
16. Masseroli M, Martucci D, Pinciroli F. GFINDer: Genome Function INtegrated Discoverer through dynamic annotation, statistical analysis, and mining. *Nucleic Acids Research* 2004 Jul; 1:32(Web Server issue); W293-300.
17. The Gene Ontology. [Online]. 1999; Disponibile su: URL: http://www.geneontology.org/
18. Pfam. [Online]. Disponibile su: URL: http://pfam.sanger.ac.uk/
19. OMIM - Online Mendelian Inheritance in Man. [Online]. Disponibile su: URL: http://www.ncbi.nlm.nih.gov/omim
20. Ontologizer, a tool for statistical analysis and visualization of high-throughout biological data using Gene Ontology. [Online]. Disponibile su: URL: http://compbio.charite.de/index.php/ontologizer2.html
21. Bauer S, Grossmann S, Vingron M, Robinson PN. Ontologizer 2.0 - A multifunctional tool for GO term enrichment analysis and data exploration. *Bioinformatics* 2008; 24(14): 1650-1651.
22. Ardagna D, Fugini MG, Pernici B, Plebani P. Sistemi informativi basati su Web. Milano: FrancoAngeli; 2006.
23. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). [Online]. 2007; Disponibile su: URL: http://www.w3.org/TR/soap12-part1/
24. Fielding RT. Architectural styles and the design of network-based software architectures. Doctoral dissertation; University of California, Irvine, 2000.

25. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. [Online]. 2007; Disponibile su: URL: http://www.w3.org/TR/wsdl20/
26. Web Application Description Language. [Online]. 2009; Disponibile su: URL: http://www.w3.org/Submission/wadl
27. Apache CXF: An Open-Source Services Framework. [Online].Disponibile su: URL: http://cxf.apache.org/
28. JSR 311: JAX-RS: The Java™ API for RESTful Web Services. [Online]. Disponibile su: URL: http://jcp.org/en/jsr/detail?id=311
29. Apache Tomcat. [Online]. 1999 Disponibile su: URL: http://tomcat.apache.org/
30. PostgreSQL:The world's most advanced open source database. [Online]. 2010 Disponibile su: URL: http://www.postgresql.org/
31. PostgreSQL 8.4.5 Documentation. Advantages of using PL/pgSQL. [Online]. 2010; Disponibile su: URL: http://www.postgresql.org/docs/8.4/interactive/plpgsql-overview.html#PLPGSQL-ADVANTAGES
32. Java. [Online]. Disponibile su: URL: http://www.oracle.com/technetwork/java/index.html
33. PostgreSQL JDBC Driver. [Online]. 2004; Disponibile su: URL: http://jdbc.postgresql.org/
34. JDOM. [Online]. Disponibile su: URL: http://www.jdom.org/
35. Document Object Model (DOM). [Online]. Disponibile su: URL: http://www.w3.org/DOM/
36. Java EE at a glance. [Online]. Disponibile su: URL: http://www.oracle.com/technetwork/java/javaee/overview/index.html
37. Garrett JJ. Ajax: a new approach to web applications. [serial online]. 2005 Feb 18; Disponibile su: URL: http://www.adaptivepath.com/ideas/essays/archives/000385.php
38. Masseroli M, Ceri S, Tettamanti L, Ghisalberti G, Campi A. Model-driven modular integration of genomic and proteomic information. *IEEE Transaction on Knowledge and Data Engineering* (sottomesso per la pubblicazione)
39. Robinson PN, Wollstein A, Böhme U, Beattie B. Ontologizing gene-expression microarray data: characterizing clusters with Gene Ontology. *Bioinformatics* 2004; 20(6):979-981.

40. Grossmann S, Bauer S, Robinson PN, Vingron M. Improved detection of overrepresentation of Gene-Ontology annotations with parent–child analysis. *Bioinformatics* 2007; 23(22):3024-3031.
41. Alexa A, Rahnenfuhrer J, Lengauer T. Improved scoring of functional groups from gene expression data by decorrelating GO graph structure. *Bioinformatics* 2006; 22(13):1600-1607.
42. Bauer S, Gagneur J, Robinson PN. GOing Bayesian: model-based gene set analysis of genome-scale data. *Nucleic Acids Research* 2010 Feb 19; 45:1-10.
43. Benjamini Y, Yekutieli D. The control of the False Discovery Rate in multiple testing under dependency. *The Annals of Statistics* 2001; 29(4):1165-1188.
44. PostgreSQL 8.4.5 Documentation. Partitioning. [Online]. 2010 Disponibile su: URL: http://www.postgresql.org/docs/8.4/static/ddl-partitioning.html
45. Martìn A, Quevedo S, Mato S. Fisher's mid-p-value arrangement in 2x2 comparative trials. *The statistical software newsletter* 1998; 29(1): 107-115.
46. Taverna Workflow Management System. [Online]. Disponibile su: URL: http://www.taverna.org.uk/
47. SeCO. The Search Computing Project. [Online]. 2009; Disponibile su: URL: http://www.search-computing.it/
48. Masseroli M, Galati O, Pinciroli F. GFINDer: genetic disease and phenotype location statistical analysis and mining of dinamically annotated gene lists. *Nucleic Acids Research* 2005; 33:W717-W723.
49. Masseroli M. Management and analysis of genomic functional and phenotypic controlled annotations to support biomedical investigation and practice. *IEEE Transactions on Information Technology in Biomedicine* 2007; 11(4): 376-385.
50. Entrez Gene. [Online]. Disponibile su: URL: http://www.ncbi.nlm.nih.gov/gene

.

Capitolo 11

Appendice

In questo capitolo sono inseriti gli schemi logici ed E-R dei database progettati non illustrati all'interno dei capitoli principali della Tesi. Inoltre vengono proposti alcuni diagrammi di flusso delle operazioni del sistema sviluppato.

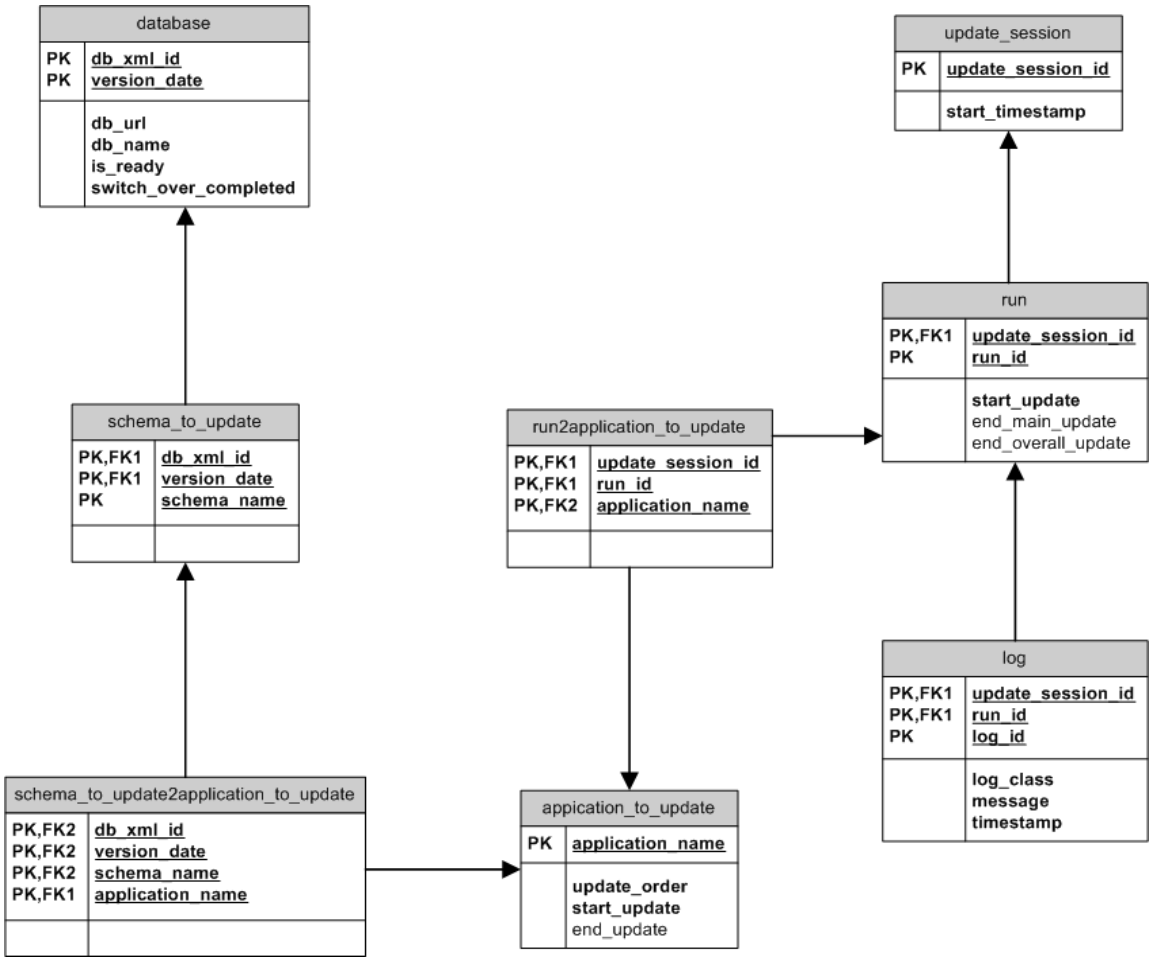
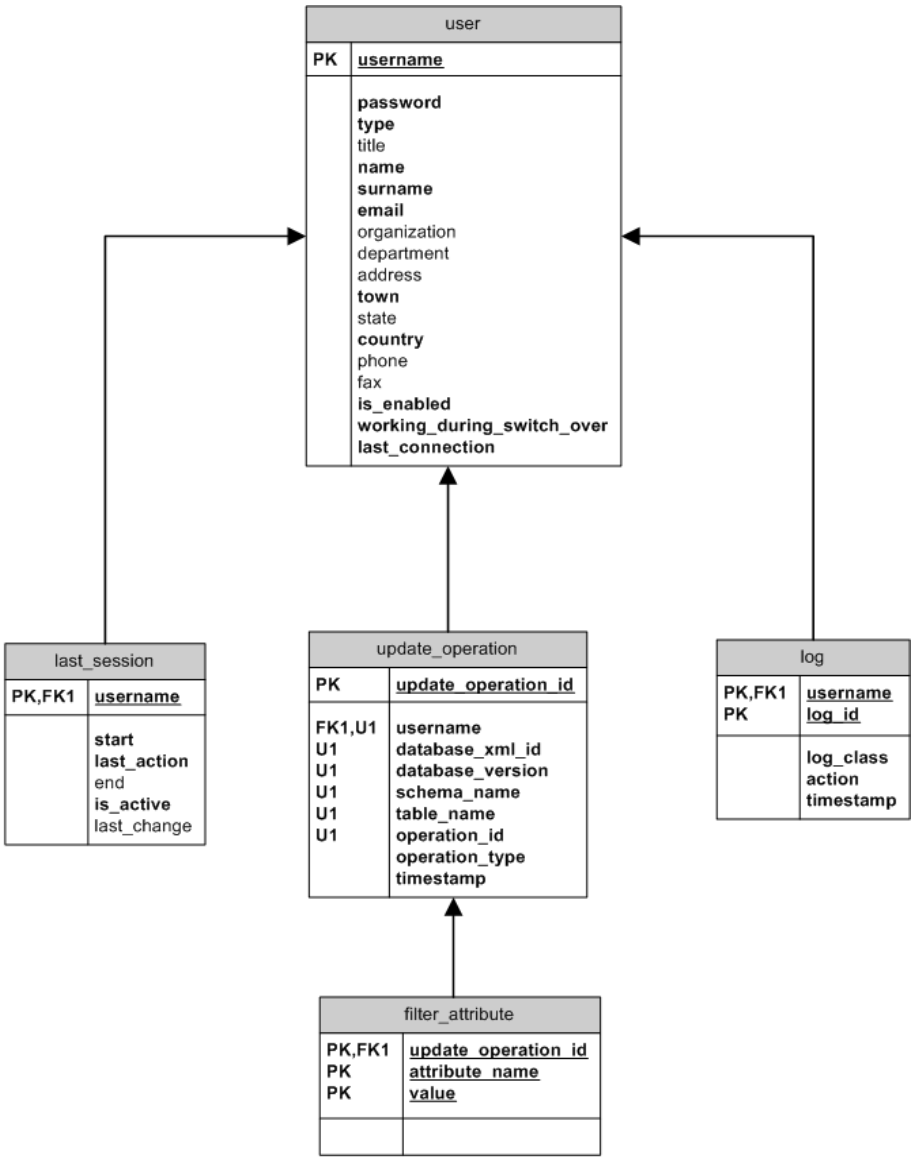


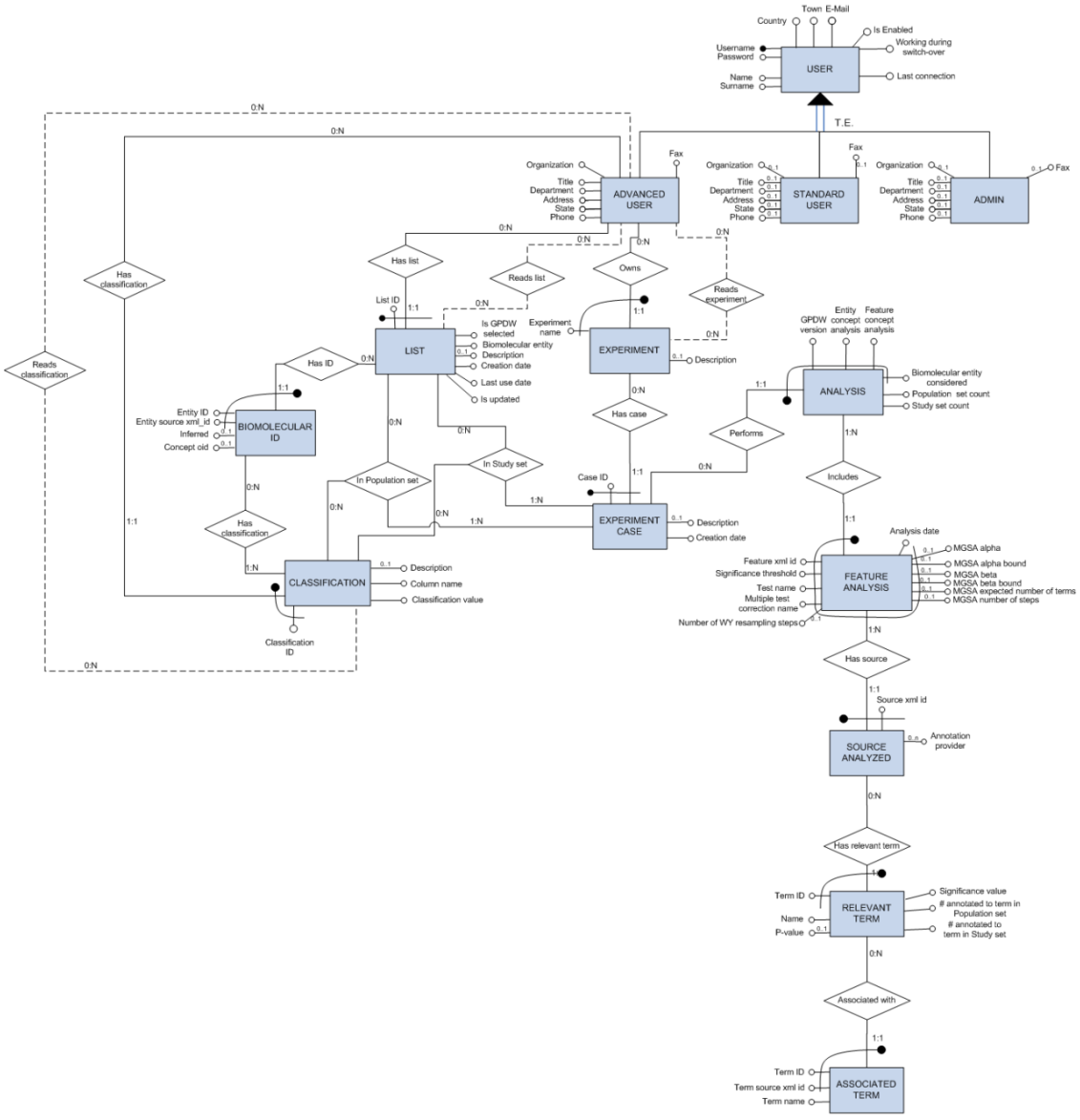
Figura 11.1. Schema logico del database GPDW_system



Application database

- The table update_operation is the result of the merge of the entity TABLE TO UPDATE and OPERATION of the E-R diagram associated
- The table filter_attribute is obtained by merging the entity FILTER ATTRIBUTE in the E-R diagram and its multivalued attribute Value

Figura 11.2. Schema logico del database del sistema realizzato



Application database diagram

- The entity USER with all its attributes will be in a separate application database with the entities LOG, LAST SESSION, TABLE TO UPDATE, OPERATION and FILTER ATTRIBUTE. In this database schema of the application present in the GPDW there will be a reduced USER entity with only the Username attribute (in this entity both standard users and advance users will be included)
- Biomolecular entity considered, in entity ANALYSIS, indicates on which biomolecular entity the annotation and the analysis is to be performed: for example, in a list there could be Dna sequence IDs, but the annotation will be made on the corresponding genes or proteins
- The attribute Multiple test correction name in the entity FEATURE ANALYSIS won't ever be null: if no correction is selected, it assumes the value "None"
- The attributes Significance threshold in the entity FEATURE ANALYSIS and Significance value in the entity RELEVANT TERM represent either the adjusted p-value (in case of standard analysis) or the marginal value (in case of MGSA analysis)
- Associations indicated with ---- have not been implemented yet

Figura 11.3. Diagramma E-R dello schema dei dati del sistema all'interno della base di conoscenza: le parti tratteggiate non sono state ancora implementate

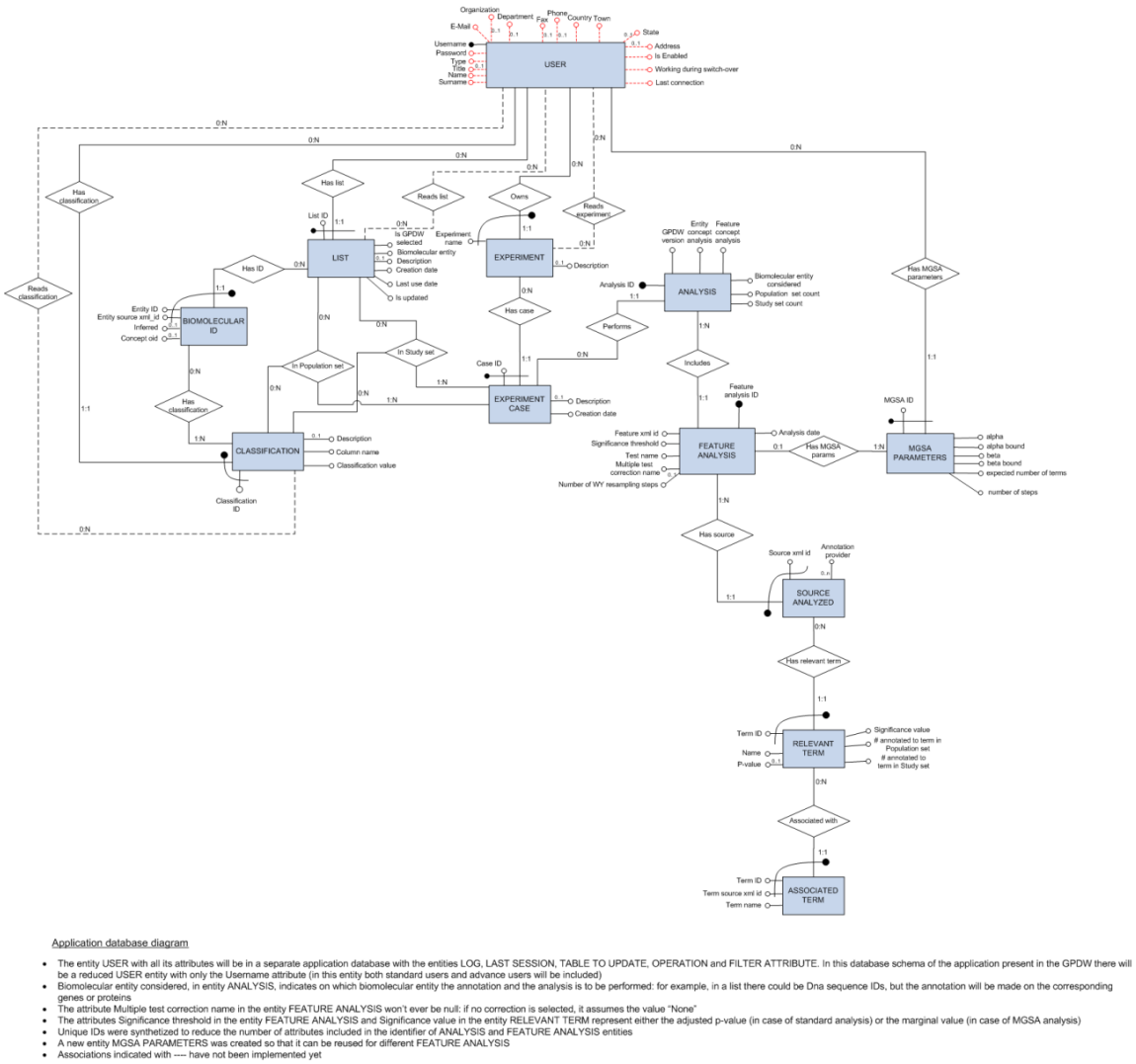
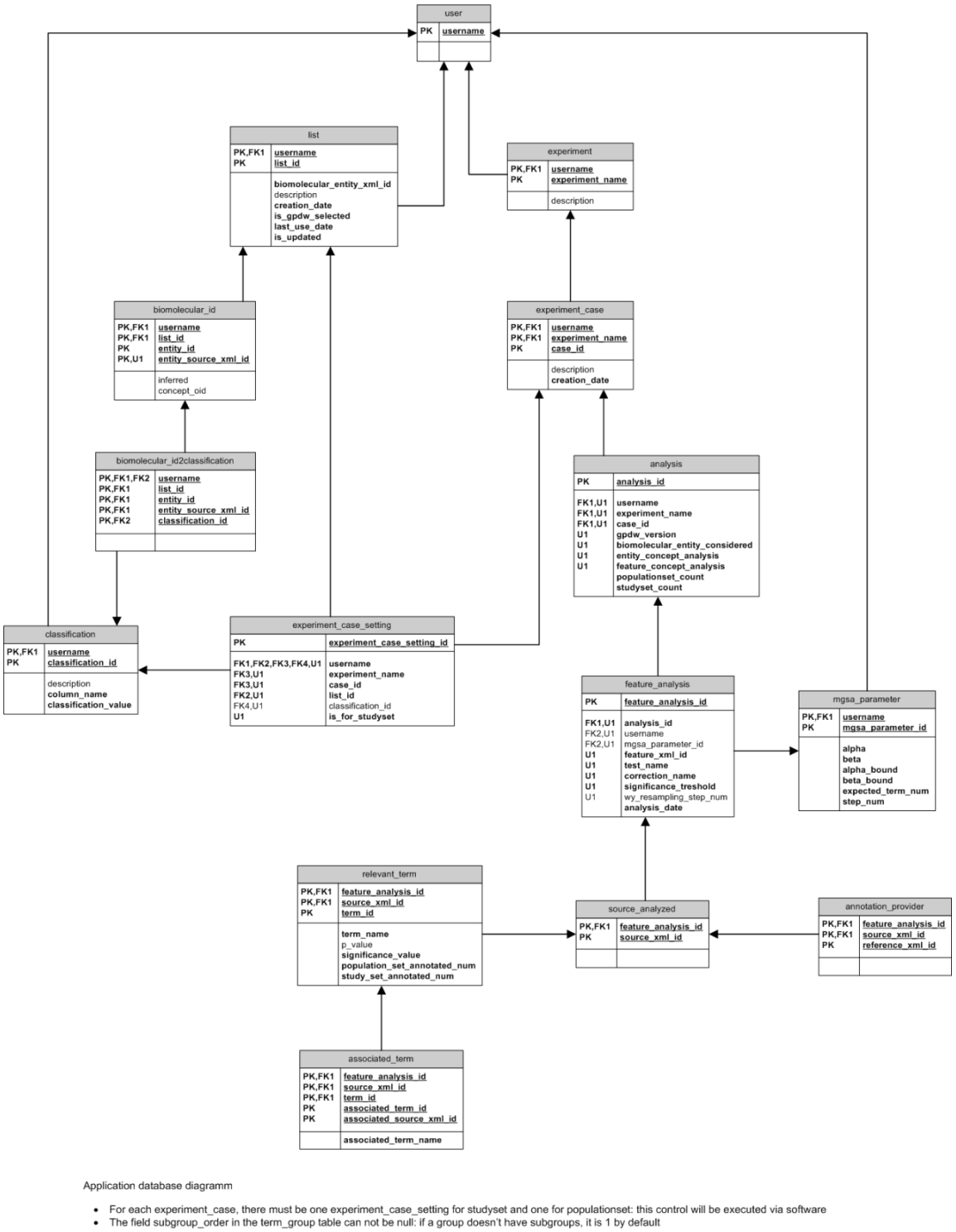


Figura 11.4. Diagramma E-R ristrutturato dello schema dei dati del sistema all'interno della base di conoscenza: le parti tratteggiate non sono state implementate, gli attributi segnalati in rosso sono inseriti per completezza ma implementati fisicamente in un altro database



Application database diagramm

- For each experiment_case, there must be one experiment_case_setting for studyset and one for populationset: this control will be executed via software
- The field subgroup_order in the term_group table can not be null: if a group doesn't have subgroups, it is 1 by default

Figura 11.5. Diagramma logico dello schema dei dati del sistema all'interno della base di conoscenza

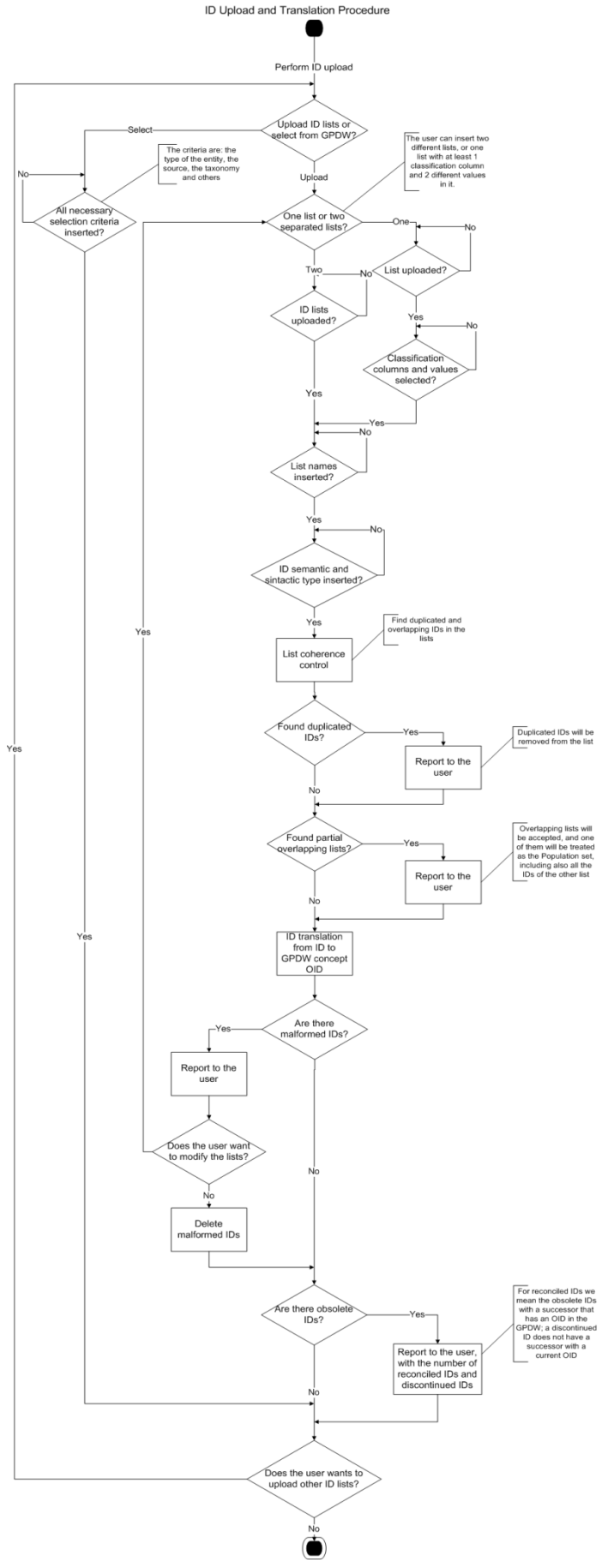


Figura 11.6. Diagramma di flusso della fase di caricamento e traduzione ID

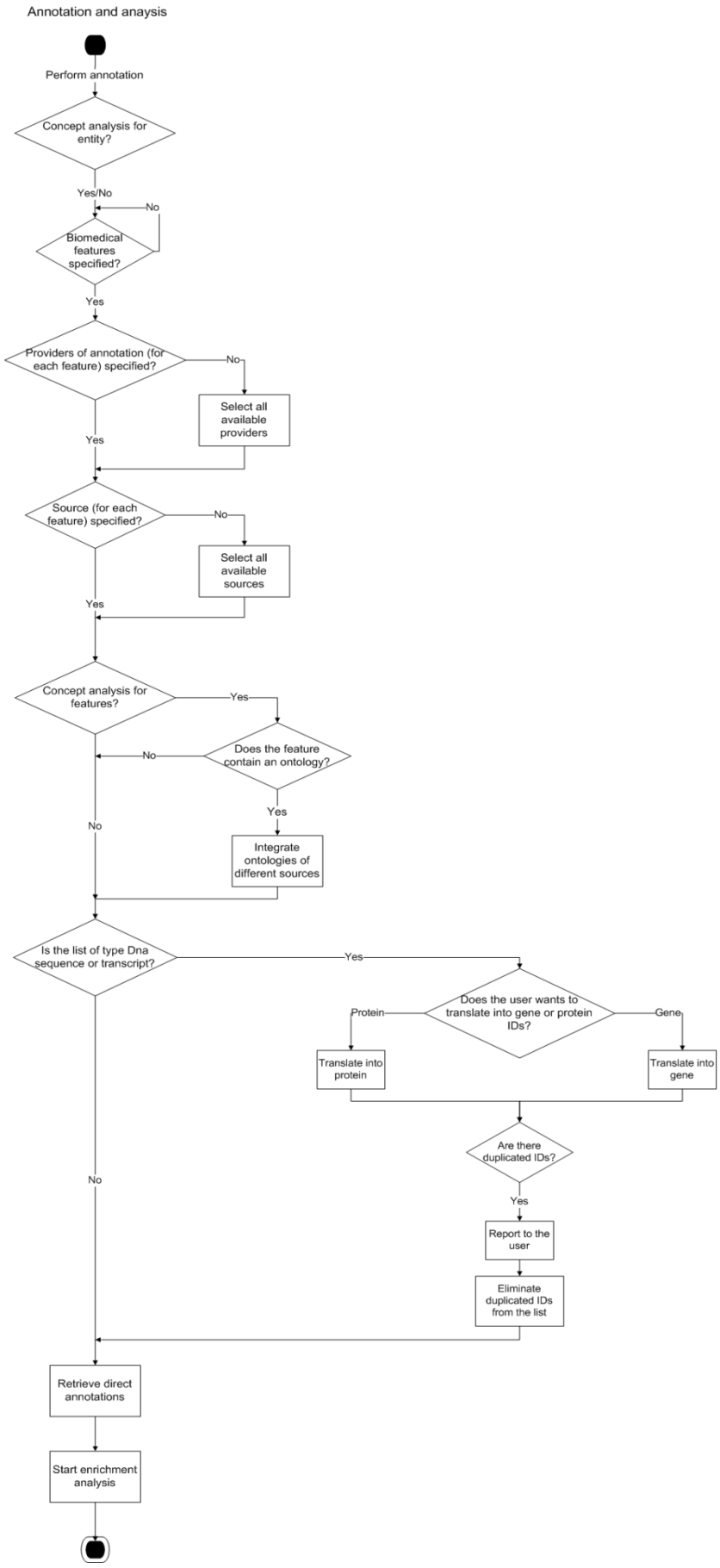


Figura 11.7. Diagramma di flusso delle operazioni preliminari all'analisi di arricchimento

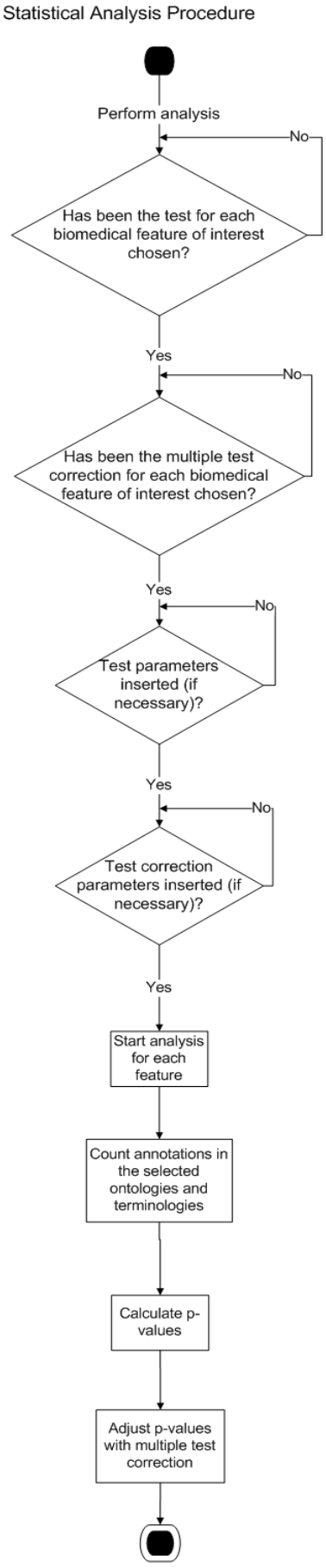


Figura 11.8. Diagramma di flusso delle operazioni di analisi statistica di annotazioni

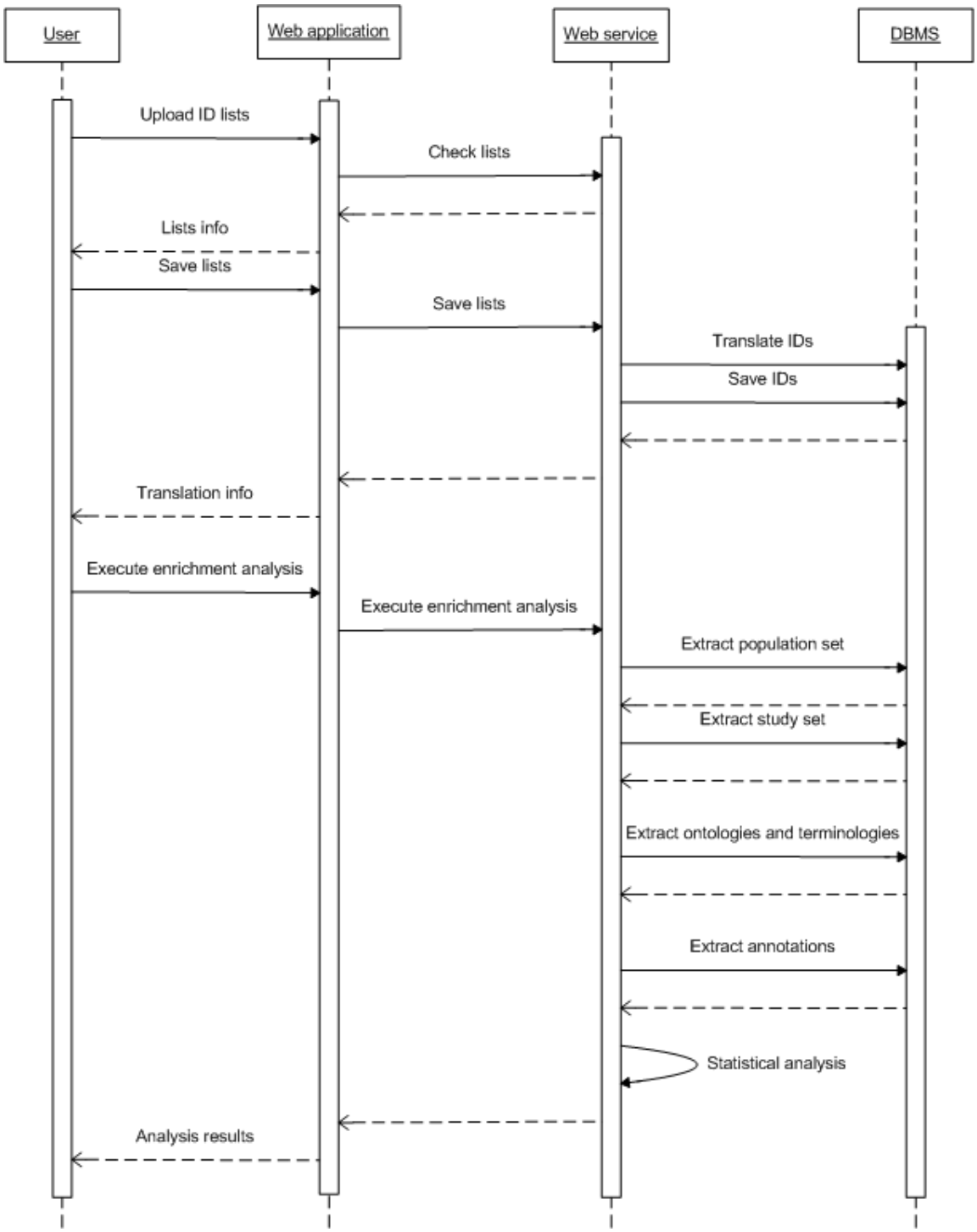


Figura 11.9. Sequence diagram che rappresenta l'interazione fra i componenti del sistema per l'esecuzione di un'analisi di arricchimento