UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

FACOLTÀ DI SCIENZE
MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN INFORMATICA

VOGCLUSTERS: una Web Application per il trattamento e l'analisi di ammassi globulari

Tesi di Laurea Sperimentale

Tutor accademico
Dott.ssa Anna Corazza

Candidato
Sabrina Checola
matr. 566/1577

Tutor aziendale
Dott. Massimo Brescia

ANNO ACCADEMICO 2008/2009

Ai miei genitori

Indice

Indice delle figure

Introduzione

L’obiettivo del progetto denominato VOGCLUSTERS è la realizzazione di una *web application* per il trattamento, archiviazione ed elaborazione di archivi astronomici relativi ad ammassi globulari. L'applicazione si propone quindi di rendere un servizio sia basato sulla semplice e veloce consultazione dell'archivio uniformato (da parte di utenti non necessariamente registrati), sia la manipolazione e integrazione con informazioni aggiuntive (da parte di utenti registrati ed autorizzati).

Per capire lo scenario in cui si colloca il presente lavoro, occorre precisare il contesto scientifico-disciplinare entro cui si inserisce la problematica affrontata.

In Astrofisica, un ammasso globulare è un insieme sferoidale di stelle che orbita come un satellite intorno al centro di una galassia. Gli ammassi globulari sono sorretti al loro interno da una forte gravità, che dà loro il tipico aspetto sferico e mantiene al loro centro una densità di stelle relativamente molto elevata. Il nome di questa categoria di oggetti deriva dal latino *globus*, che significa "globo", "sfera". Talvolta ci si riferisce a questi oggetti semplicemente con l'appellativo *globulare*. Gli ammassi globulari sono in genere composti da centinaia di migliaia di stelle vecchie, le stesse che compongono il nucleo, noto come *bulge*, di una galassia spirale, ma confinate in pochi parsec cubici. Gli ammassi globulari sono piuttosto numerosi: se ne conoscono attualmente poco più di un centinaio attorno alla Via Lattea, con forse altri 10-20 da scoprire, essendo nascosti all'osservazione da Terra dalle polveri interstellari che oscurano la vista in direzione del centro galattico; pare che le galassie più grandi possano averne un numero nettamente superiore (la Galassia di Andromeda potrebbe averne fino a 500). Alcune galassie ellittiche giganti (come M87) ne contano fino a 10.000. Questi oggetti sono considerati parte dell'*alone* delle galassie, orbitando attorno ai centri di queste a distanze fino a 131.000 anni luce. Attualmente la formazione di un ammasso globulare resta un fenomeno piuttosto misterioso. Gli studiosi non sono sicuri se le stelle si sono formate in una singola generazione, o si estendono per diverse generazioni in periodi di diverse centinaia di milioni di anni. Questo periodo di formazione stellare è tuttavia relativamente breve se paragonato all'età di molti ammassi. Le osservazioni mostrano che la formazione delle stelle degli ammassi globulari avviene innanzitutto in regioni dove questo fenomeno è molto elevato e dove il mezzo interstellare ha una densità maggiore rispetto alle regioni normali di formazione stellare. La formazione dei globulari avviene principalmente nelle regioni dette *starbust* e nelle galassie interagenti.

L’indagine scientifica di tali oggetti si basa sulla possibilità di confrontare i dati tabellari esistenti (archivi o database di ammassi globulari) e sull’interoperabilità tra le informazioni presenti in

questi archivi distribuiti, possibilmente realizzata a seguito di applicazioni dedicate, in grado di rappresentare le varie tipologie d'informazione disponibili (parametri fisici, riferimenti e note bibliografiche, sovrapposizione e analisi statistica di dati, immagini, spettri etc.). Il lavoro oggetto di questa tesi sperimentale si inserisce proprio in questo contesto, acquisendo di fatto una valenza scientifica oggettiva, oltre che ponendosi, come sarà reso chiaro nel seguito, in un preciso contesto multidisciplinare, come ottimo esempio di utilizzo dello stato dell'arte dell'ICT al servizio della speculazione astrofisica.

Da un punto di vista più vicino agli aspetti dell'ICT (*Information & Communication Technology*) in ambito astrofisico, al pari di altri contesti scientifici (fisica delle alte energie, fisica nucleare, biologia, scienze della terra etc.), lo sviluppo della tecnologia per la strumentazione (nel caso specifico telescopi, strumenti di piano focale, rivelatori a grande campo, architetture HPC per le simulazioni numeriche) ha generato un "effetto tsunami" sulla quantità di dati ottenuti a seguito di osservazioni (da terra e dallo spazio) e/o simulazioni (modelli cosmologici, *template* di oggetti in evoluzione, piano fondamentale, etc.). A partire dal 2003, la comunità astrofisica si è posta l'annoso problema della memorizzazione ed analisi di enormi moli di dati (MDS o *Massive Data Sets*), caratterizzati da un alto grado di delocalizzazione dei database (DB) e da una rappresentazione multi-banda e multi-epoca sfociante in uno spazio dei parametri a N dimensioni, con N >> 100. La quantità di dati mediamente ricavabile da una sola osservazione con telescopi di moderna generazione è quantificabile in decine di TB (TeraByte) al giorno. L'esigenza di conciliare la visione 3D dell'astronomo rispetto alle centinaia di dimensioni dello spazio dei parametri, nonché la complessità di compiere scienza su dati di tali dimensioni, ha provocato il fiorire di iniziative volte a creare degli standard di rappresentazione, ontologie per la classificazione delle categorie di dati e strumenti efficienti di *information retrieval* su DB di enormi dimensioni. Tale operazione ha portato alla costituzione dell'IVOA (*International Virtual Observatory Alliance*), consorzio mondiale per la costituzione di un'infrastruttura unica in grado di rappresentare in modo universale i dati provenienti da osservazioni/simulazioni astronomiche a grande campo. Il paradigma del VO è molto interessante in quanto permette di reperire in modo trasparente le informazioni e i dati disponibili in formato digitale nei centri dati collegati in rete. L'IVOA definisce gli standard per l'accesso ai dati e ai servizi disponibili all'interno delle singole comunità di Virtual Observatories (VObs). A fine settembre 2007, il Comitato Esecutivo IVOA ha approvato la prima versione degli standard di base VObs, dopo un periodo di test estensivo e con l'implementazione di prototipi. Molti strumenti software di gestione ed elaborazione sono adesso in grado di trattare dati in modo compatibile con il VObs e molti altri saranno disponibili in futuro. Tuttavia il legame tra VObs e sistemi di esplorazione scientifica dei dati è attualmente ancora in fase embrionale. In effetti, IVOA finora ha concentrato i suoi sforzi di standardizzazione principalmente sui dati osservativi e simulazioni, creando di fatto un gap con le tecniche, tutt'altro

che standardizzate, fruibili per la successiva indagine e manipolazione dei dati (KDD, Knowledge Discovery in Databases).

Il Progetto DAME[1] (*Data Mining & Exploration*), entro cui si inserisce l'applicazione VOGCLUSTERS, nasce proprio come strumento per colmare tale lacuna, con lo scopo cioè di fornire una suite di *Data Mining* (DM) orientata al web, in grado di permettere ai suoi utilizzatori di effettuare esperimenti di esplorazione ed analisi di dati astronomici con un alto grado di complessità, sfruttando il paradigma VO per la loro rappresentazione standardizzata ed omogenea.

Per tale motivo, la suite DAME utilizzerà una potente infrastruttura computazionale, realizzata dal progetto SCoPE (Sistema Cooperativo per Esperimenti scientifici ad alte prestazioni), basata su paradigma Grid e sulle più moderne tecnologie per il calcolo distribuito, messa a disposizione dall' Università di Napoli Federico II per la Ricerca di base. Il Progetto SCoPE infatti prevede lo sviluppo di codici innovativi per applicazioni in diversi settori della ricerca scientifica di base tra i quali:

- Astrofisica (dove appunto si colloca il Progetto DAME)
- Fisica Subnucleare
- Matematica Numerica
- Scienze della vita
- Informatica
- Elettromagnetismo e telecomunicazioni

L'infrastruttura di SCoPE consiste nella creazione di una Grid Metropolitana capace di unire dipartimenti e struttura di ricerca afferenti o in collaborazione con la Federico II, situate in varie zone della città di Napoli[2].

Partnership del progetto DAME:

- Dipartimento di Fisica (sezione Astrofisica) – Università degli studi di Napoli Federico II
- INAF – Osservatorio Astronomico di Capodimonte
- California Institute of Technology, Pasadena – USA

Collaborazioni:

- VOTECH (Virtual Observatory Technological Infrastructures)
- S.Co.P.E.
- INAF - Osservatorio Astronomico di Trieste
- MIUR (Ministero dell'Istruzione, dell'Università e della Ricerca)

[1] http://voneural.na.infn.it
[2] http://www.scope.unina.it

- EURO-VO (laboratorio di osservazione virtuale europeo)

La *web application* oggetto di questo lavoro di tesi è da intendersi quindi come un strumento integrato nel progetto DAME, sebbene indipendente dal punto di vista dell'utilizzo finale, nel senso di essere specializzato per il trattamento di un preciso problema astrofisico. In particolare, l'integrazione in DAME riguarda molte delle soluzioni tecnologiche adottate, facenti riferimento ai requisiti legati alla progettazione e realizzazione della Suite DAME.

Per quanto riguarda gli aspetti inerenti la sicurezza della suite, poiché l'applicazione condivide l'archivio utenti della Suite DAME, ed essendo installata su una piattaforma GRID pre-esistente, i livelli necessari di sicurezza ed integrità dell'informazione sono garantiti. Infatti, unendo la policy dell'infrastruttura GRID del progetto S.Co.P.E. circa la sicurezza e rintracciabilità dei processi e degli accessi mediati da un apposito proxy system, alla policy di registrazione ed accounting degli utenti DAME , VOGCLUSTERS detiene indirettamente tutti i necessari livelli di sicurezza. Infine, come in DAME, ogni utente registrato ed autenticato non potrà accedere ad alcuna informazione riservata degli altri utenti, potendo visualizzare solamente informazioni relative ai dati scientifici condivisi. Ciò ad ulteriore garanzia del rispetto delle normative sulla privacy.

In stretta analogia con il metodo ingegneristico adottato per la suite DAME, anche il processo di sviluppo dell'applicazione VOGCLUSTERS è stato caratterizzato dall'analisi dei requisiti, seguita dall'indagine progettuale ed infine dall'implementazione. Per ogni fase, il lavoro ha prodotto documentazione tecnica di progetto secondo gli standard IEEE, dal P.D.D. (*Project Description Document*), all'S.R.S. (*Software Requirement Specifications*), al S.D.D. (*Software Design Description*), fino alla documentazione di test.

Il capitolo 1 espone le principali caratteristiche dell'architettura del progetto DAME, della web application VOGCLUSTERS e del sistema di interfacciamento e scambio d'informazioni tra le due applicazioni.

Il capitolo 2 è dedicato ad una dettagliata descrizione di tutte le tecnologie ICT impiegate per implementare i componenti dell'applicazione VOGCLUSTERS, sulla base dei requisiti scientifici e funzionali di progetto. Ampia trattazione è anche dedicata allo schema ed ai protocolli di comunicazione interna tra i componenti dell'applicazione.

Il capitolo 3 contiene la descrizione delle soluzioni implementate per la base di dati relativa agli ammassi globulari per il flusso I/O tra i componenti e per l'interfacciamento con l'utente finale.

Il capitolo 4 riporta i risultati del test dell'applicazione.

Infine nel capitolo 5 vengono analizzati i possibili sviluppi futuri.

Capitolo 1

Architettura del sistema

La progettazione di un'infrastruttura software, fase primaria di un progetto scientifico e/o tecnologico, enuclea diversi aspetti: (i) si concentra sulla struttura del sistema e sulle relazioni esistenti fra le parti costituenti; (ii) identifica le operazioni logiche da svolgere; (iii) individua le modalità con cui il sistema può e deve interagire con il mondo esterno. Il risultato della progettazione è la definizione dell'architettura del sistema, intendendo con questo termine l'organizzazione strutturale del sistema stesso, che comprende i suoi componenti software, le proprietà visibili esternamente di ciascuno di essi (l'interfaccia dei componenti) e le relazioni fra le parti.

Bass, Clements e Kazman (Bass, 2003) definiscono in questo modo l'architettura del software:

> *L'architettura del software di un programma o di un sistema di calcolo è costituita dalle strutture del sistema, che comprendono i componenti software, le loro proprietà visibili e le relazioni fra di essi.*

Nello sviluppo delle applicazioni software, è possibile descrivere l'architettura del sistema utilizzando uno fra i diversi paradigmi a disposizione, ma in linea generale, la forma più nota ed usata riguarda l'architettura a "livelli" (*Layered Application Architecture*). Questo stile di architettura prevede la strutturazione logica di un sistema software in strati sovrapposti (*layer*), nettamente distinti e in grado di comunicare tra di loro secondo regole gerarchiche ben precise. Nella maggior parte dei casi i livelli logici che si distinguono sono tre:

- ***Data Access Layer*** **(DAL)**: è lo strato di accesso ai dati, dove sono gestite le interazioni con il sistema di persistenza delle informazioni (generalmente un database). Il DAL permette al Business Logic Layer di reperire le informazioni sulle quali effettuare le elaborazioni.

- ***Business Logic Layer*** **(BLL)**: è lo strato di business, dove sono presenti i servizi applicativi.

- ***User Interface*** **(UI)**: è lo strato di presentazione, il cui scopo primario consiste nel gestire l'interazione del sistema con il mondo esterno, in particolare con gli utenti. Include gli strumenti per la visualizzazione e l'inserimento dei dati, i controlli, dai più semplici ai più complessi, e i meccanismi per intercettare e trattare opportunamente gli eventi generati in funzione delle azioni svolte dagli utenti.

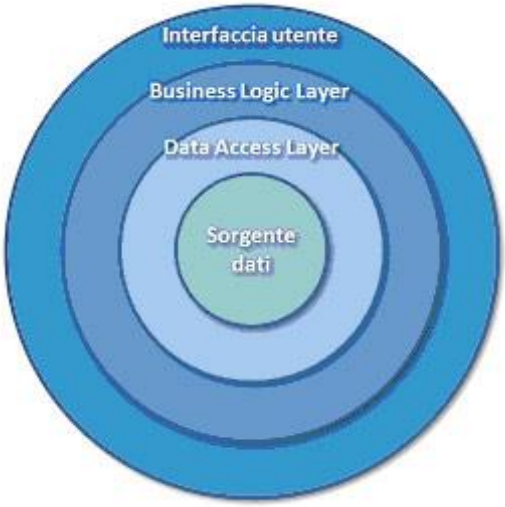

Figura 1: Architettura a livelli

La web application, oggetto di questo lavoro, si colloca nel contesto preesistente della suite DAME. Nei paragrafi che seguiranno verrà mostrato lo schema generale dell'architettura, mostrando come i due sistemi (la suite DAME e VOGCLUSTERS) comunicano tra di loro scambiandosi informazioni e servizi.

1.1 ARCHITETTURA DELLA SUITE DAME

DAME è un progetto volto a sviluppare strumenti ICT per il *data mining* scientifico. I dati scientifici, di natura astrofisica e caratterizzati da enormi *dataset*, sono raccolti e memorizzati mediante diversificati e spesso incompatibili archivi di dati, localizzati su scala mondiale nei diversi *data center* astronomici.

Il Progetto DAME mira a creare una singola "*e-infrastructure*" distribuita per il *data mining*, nonché per l'esplorazione e la visualizzazione di MDS. Essa deve fornire un accesso integrato ai dati raccolti da strumenti, esperimenti, e comunità scientifiche spesso eterogenee, al fine di essere in grado di correlare e migliorare la loro utilità scientifica e l'interoperabilità.

Il progetto consiste nello sviluppo di una "*Data Mining Suite*" che fornirà alle comunità astronomiche la possibilità di lavorare su enormi *dataset* in un ambiente di calcolo distribuito rispettando gli standard internazionali e i requisiti IVOA.

In generale per *Data Mining* (letteralmente: *estrazione di dati*) si intende estrarre conoscenza, non nota a priori, a partire da grandi quantità di dati intrinsecamente dotati di notevoli quantità di *noise*.

Oggi il *data mining* ha una duplice valenza:

- **Estrazione**, cioè estrarre informazione implicita o nascosta da dati già strutturati, con tecniche analitiche, per renderla disponibile e direttamente utilizzabile;

- **Esplorazione ed analisi**, eseguita in modo automatico o semiautomatico, su grandi quantità di dati allo scopo di identificare pattern significativi;

Nella ricerca scientifica molti fattori hanno contribuito allo sviluppo del *data mining* come il *data storage* e l'introduzione di nuovi metodi e tecniche di analisi basate sull'apprendimento automatico e riconoscimento di *pattern*.

Le tecniche di *data mining* sono fondate su specifici algoritmi e hanno lo scopo principale di trovare nuovi *pattern*, che a loro volta, potrebbero essere il punto di partenza per la scoperta di nuove relazioni di tipo causale tra fenomeni, e quindi, effettuare delle previsioni e classificazioni su nuovi insiemi di dati.

Nonostante questi progressi, rimane di assoluta rilevanza il problema di trovare metodi adeguati per l'esplorazione e l'analisi di grandi insiemi di dati astronomici.

L'accento è posto su metodi che colmino il divario tra accurate rappresentazione ed estrazioni di dati, e anche il divario tra le potenzialità della tecnologia attuale e gli utenti.

DAME nasce quindi come applicazione adatta a lavorare su MDS e strutturata sottoforma di una suite *web-oriented* che rispetta gli standard di rappresentazione dei dati della comunità internazionale VO.

Al fine di effettuare esperimenti di data mining e di esplorazione di dati, la suite DAME è stata progettata con una strategia top-down, a partire dalla tassonomia delle varie strategie di data mining e di ricerca dei dati, che sono associate a specifici algoritmi e metodi di lavorazione.

Nella prima versione, la suite offre strumenti per le seguenti funzionalità di esplorazione dei dati:

- Classificazione
- Regressione

Le tecnologie che caratterizzano l'intera suite possono essere riassunte come segue:

- OOP (*Object Oriented Programming*) & UML (*Unified Modeling Language*);
- Standard e protocolli interni basati su XML e VOTable;
- Architettura Java Web Application;
- Database utilizzato per dati di I/O degli esperimenti, registrazione e autenticazione degli utenti;
- *Web-based* user I/O (GWT e Smart-GWT);
- Interfaccia utente basata su servizi web e applicazioni web (*web-tools*);

Caratteristiche principali:

- Espandibile tramite integrazione o modifica di plugin di modelli (librerie di nuovi modelli) non ancora supportati dalla suite;
- Indipendente dall'Hardware attraverso la piattaforma "driver" che permette alla suite di essere utilizzabile sia in modalità "stand alone" che in un'infrastruttura di calcolo distribuito (Grid, Cloud, HPC);
- Sessione di lavoro interattiva e/o asincrona durante l'intera fase del lancio dell'esperimento. Interattiva perché l'utente vuole monitorare/valutare/terminare l'esperimento in fase di lancio, in modo tale da modificare alcuni parametri e rilanciarlo. Asincrona nel senso che l'utente non vuole tenere traccia dell'esperimento in fase di lancio, chiudendo la connessione dopo aver inserito tutti i parametri e aspettando i risultati (ad es. attraverso e-mail);

Nella suite, un esperimento di data mining è formalizzato dall'associazione tra funzionalità e modello (algoritmo) di analisi. Le funzionalità previste (ad es. le su citate classificazione e regressione) individuano l'ambito scientifico-teorico dell'esperimento e sono implementate in modo da mantenere un alto livello di astrazione rispetto all'esperimento singolo, realizzato dall'utente finale. I modelli o algoritmi di data mining ne definiscono invece l'ambito tecnico-applicativo, completando quindi tutti gli aspetti necessari ad eseguire un esperimento di data mining. Tale costrutto progettuale si concretizza nell'atto di specializzazione dell'esperimento, attraverso il setup dei parametri della coppia funzionalità-modello. Tale fase può essere opzionalmente evitata dall'utente finale, lasciando le impostazioni di default, o accuratamente effettuata dall'utente più esperto tramite le risorse del componente Front-End (Interfaccia Utente) della Suite.

In questo scenario, la corrispondenza uno a uno tra funzionalità e modelli di *data mining* implementati dalla suite acquisisce un ruolo chiave nell'infrastruttura.

È previsto che la suite DAME sia espandibile, mediante aggiunta di nuove funzionalità e di ulteriori modelli di *data analysis* e *data mining*, senza richiedere una modifica strutturale dei

componenti software, ma semplicemente tramite registrazione semi-automatica, utilizzando appositi strumenti di manutenzione[3].

La Suite è basata su 5 componenti principali:

- Front End (FE)
- Framework (FW)
- Registry&DB (REDB)
- Driver (DR)
- Data Mining Models (DMM)

Il seguente schema mostra il diagramma dei componenti dell'intera suite DAME.

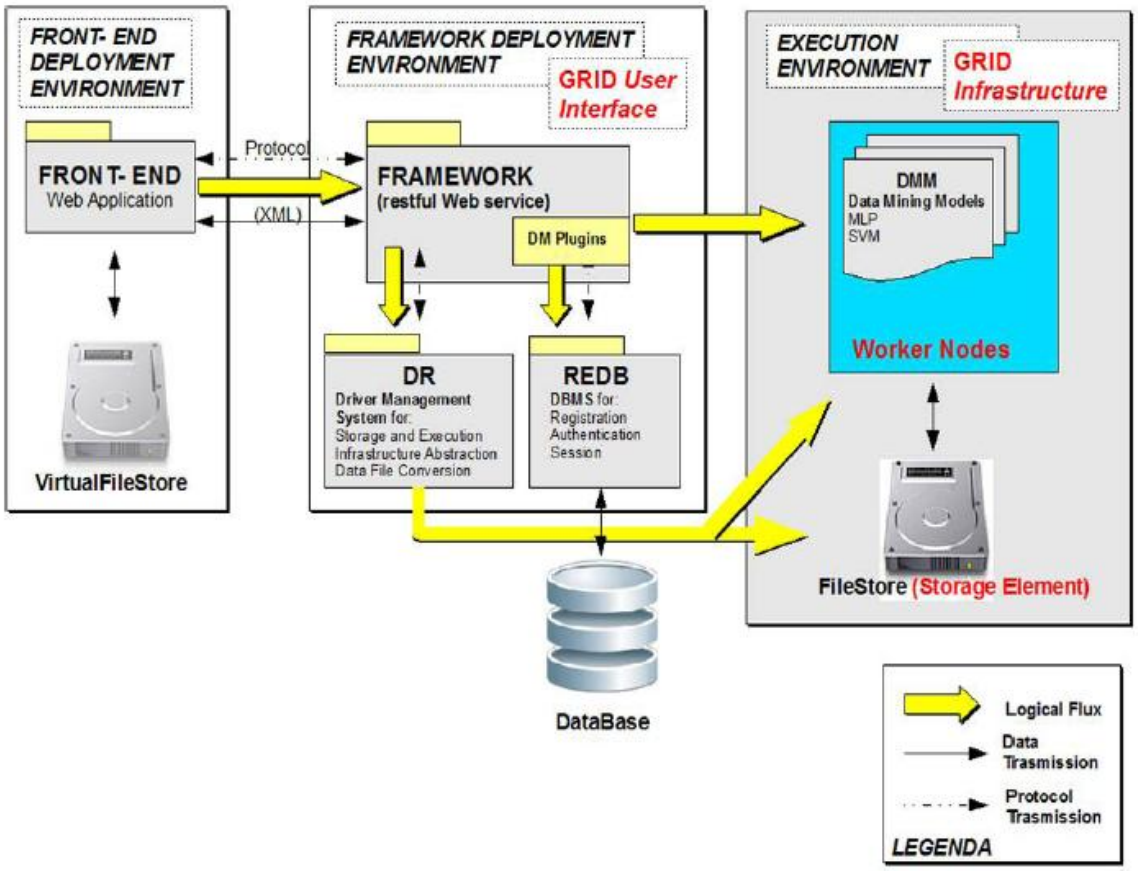

Figura 2: Diagramma dei componenti della Suite DAME

Il componente Front-End comprende la GUI (*Graphical User Interface*) della Suite.

[3] In particolare esiste uno strumento software, denominato DMPlugin, dotato di un WEB application, con cui l'utente finale esperto è in grado di introdurre nuove funzionalità e modelli.

Esso ha il compito di interagire direttamente con l'utente finale. Al suo interno è presente la logica che consente di generare le pagine WEB dinamiche che verranno utilizzate dagli utenti per interfacciarsi con l'applicazione, utilizzare i modelli, e facilitare esperimenti scientifici.

Ogni utente, prima di poter utilizzare le funzionalità della suite, deve effettuare una fase di registrazione, creandosi un account personale. L'interfaccia prevede una procedura di autenticazione (login) che reindirizza l'utente in un ambiente di lavoro personale (cartella di lavoro o *workspace*) dove verranno memorizzati sia i dati di input per il lancio di un esperimento caricabili dall'utente stesso, sia i dati di output ottenuti dall'esperimento in esecuzione. Tali dati vengono memorizzati in un VFS (*Virtual File Store*) che rappresenta un'astrazione del *File Store* fisico. Vedremo che la categorizzazione dell'utenza è uno degli aspetti che lega la Suite DAME all'applicazione VOGCLUSTERS.

Il Front-End comunica solo con il Framework tramite documenti di tipo XML (*eXtensible Markup Language*).

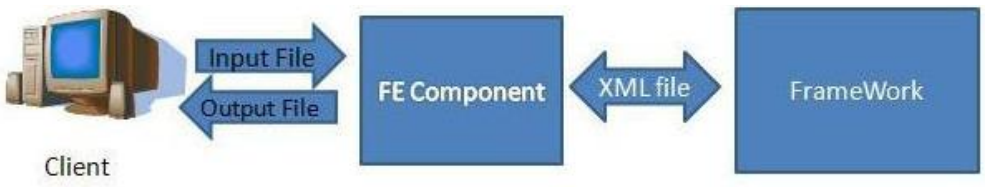

Figura 3: Comunicazione tra Client FE e FW

Il Framework rappresenta il nucleo della suite DAME ed è responsabile del coordinamento di tutti i componenti della suite. Esso ha 2 compiti principali:

- Rispondere alle richieste del Front-End, recuperando le informazioni dal DBMS (DataBase Management System);
- Fornire un servizio agli amministratori della suite, presentando un'interfaccia di amministrazione, che permetta di installare nuove funzionalità ed effettuare operazioni statistiche sul sistema;

Il componente Registry & DataBase (REDB) ha il compito principale di memorizzare tutto ciò che avviene all'interno della suite. Infatti, come si può dedurre dal nome esso è:

- **Registry**, nel senso che contiene tutte le informazioni relative agli utenti registrati e gli account, le loro relative sessioni e esperimenti;
- **DataBase,** perché contiene tutte le informazioni circa i dati di input/output di ogni esperimento, oltre che i dati temporanei e finali provenienti da processi (esperimenti) lanciati dall'utente ancora in fase di esecuzione;

Il Database è modellato con un classico sistema entità-relazione, nel quale il DBMS permette un'efficiente navigazione all'interno delle informazioni memorizzate tramite *query* standard o specifiche. Sia gli utenti che i componenti possono accedere alla base di dati in modo sicuro e protetto.

Il REDB è basato sulle seguenti tecnologie:

- MySQL DBMS Server
- Il connettore JDBC
- API di accesso ai dati

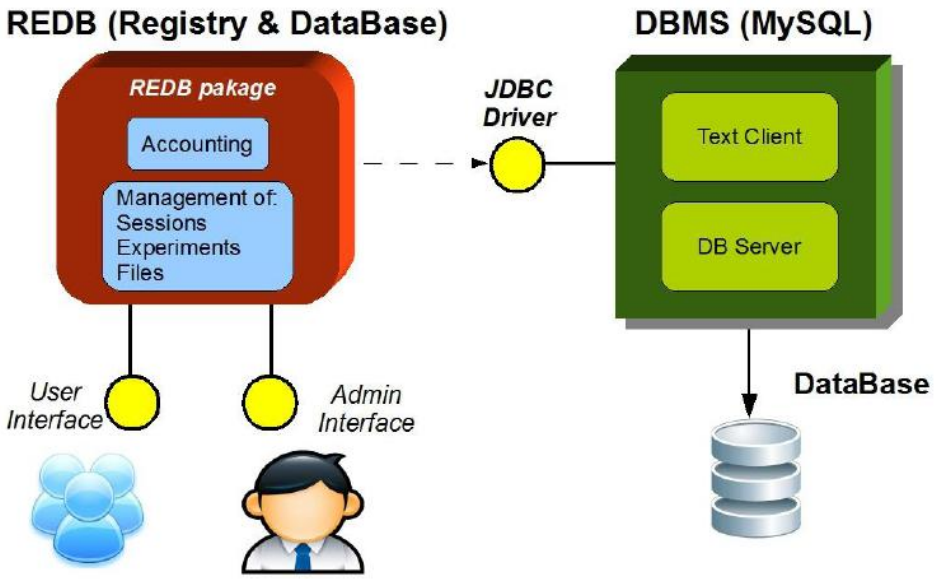

Figura 4: Comunicazione tra REDB e DBMS

Naturalmente, per ovvi motivi di compatibilità e affinità progettuale, il DBMS dell'applicazione VOGCLUSTERS è basata su uno schema e tecnologia analoghe a quella appena descritta.

Il Framework si interfaccia sempre con il componente Registry & DataBase per garantire l'integrità dei dati e la consistenza delle informazioni memorizzate dagli utenti durante la configurazione di un esperimento e durante operazioni di caricamento o modifica di file.

Il componente Driver, è stato progettato per separare i requisiti funzionali del Framework dalla sua implementazione. In altre parole, il Driver ha l'utilità di astrarre il Framework dalla piattaforma (singolo PC o infrastruttura distribuita) su cui esso adopera.

Infine il componente DMM (Data Mining Model) è strutturato sottoforma di API (*Application Programming Interface*) specializzate per implementare ed eseguire i vari modelli ed algoritmi di *data mining* previsti nella Suite.

1.2 ARCHITETTURA DI VOGCLUSTERS

Di seguito viene mostrato il diagramma delle componenti che costituiscono l'architettura della web application (Figura 5). Maggiori dettagli su ognuna di queste componenti verranno forniti nel capitolo 3. Questo paragrafo ha il solo scopo di illustrare la struttura generale del sistema, con particolare riferimento alla comunicazione tra la web application realizzata e il sistema DAME preesistente.

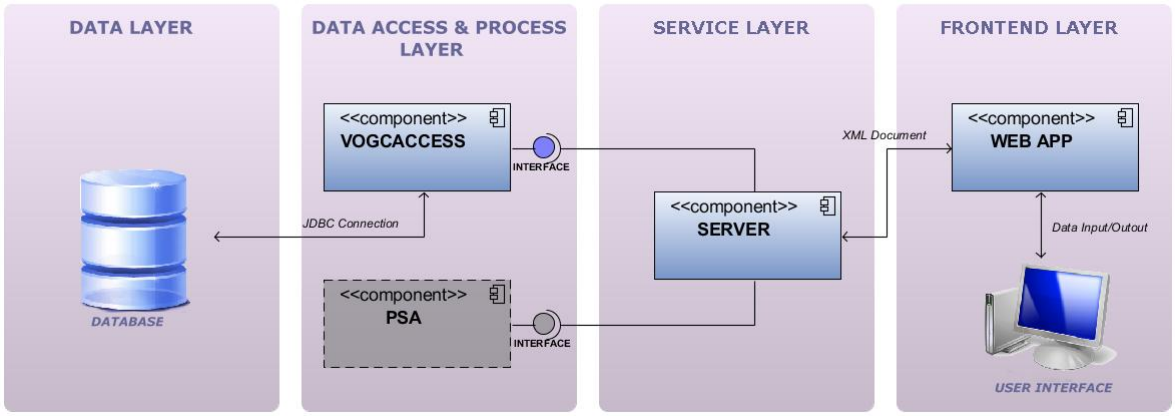

Figura 5: Architettura di VOGCLUSTERS

Analizziamo singolarmente ogni strato dell'architettura mostrata:

- **Data Layer**: questo strato ha il compito di gestire la persistenza dei dati. Al suo interno è stato collocato il Database.
- **Data Access and Process Layer**: in questo strato sono state collocate le componenti VOGCACCESS e PSA (Plotter Statistico e Analitico) che si occupano dell'accesso ai dati permanenti e al trattamento analitico e statistico degli stessi. In particolare, la componente VOGCACCESS si occupa del salvataggio e del recupero dei dati persistenti, nonché della comunicazione con il Data Layer.

  L'altra componente PSA, progettata ma non implementata nella prima release dell'applicazione, si occuperà della realizzazione di grafici statistici e analitici su particolari dati di input forniti dall'utente.

  Entrambe le componenti forniscono apposite interfacce per la comunicazione con le componenti esterne.

- **Service Layer**: è stato così definito per enfatizzarne il suo ruolo di fornitore di servizi all'interno del sistema. Infatti è lo strato che raccoglie i servizi che il sistema mette a disposizione del Frontend Layer fornendo a quest'ultimo sia l'accesso al database locale che al database della suite DAME.

- **Frontend Layer**: contiene tutta la logica del sistema che si occupa dell'interazione con l'utente finale. Al suo interno vengono mostrate due componenti: WEB APP e USER INTERFACE. La prima è la parte della web application che ha il compito di fruire delle funzionalità messe a disposizione dal Service Layer e di fornire all'utente finale le pagine dinamiche che verranno poi mostrate attraverso la USER INTERFACE.

Gli oggetti principali trattati dalla web application, sono gli ammassi globulari. L'utente che accede a VOGCLUSTERS, ha la possibilità di visualizzare la lista di tutti gli ammassi globulari (galattici ed extragalattici) registrati nel database locale.

Ogni ammasso globulare ha una serie di parametri osservativi ad esso associati ed a cui possono corrispondere diversi valori, frutto di successive evoluzioni della ricerca nel settore. L'utente avrà la possibilità, oltre che di visionare i dati, di integrarli con informazioni proprie (nuovi valori, commenti, immagini, diagrammi, etc.).

Esistono tre tipologie di utente che possono accedere al sistema: l'*amministratore DAME*, l'*utente DAME abilitato* e l'*utente generico*. Questi si distingueranno essenzialmente in base alla tipologia di manipolazione/integrazione di dati e di richieste all'interno di VOGCLUSTERS. L'applicazione prevede quindi un sistema di abilitazione degli utenti DAME. Ovviamente per poter passare da utente generico ad utente DAME abilitato è necessaria una registrazione all'interno della suite DAME.

1.3 COMUNICAZIONE FRA DAME E VOGCLUSTERS

La comunicazione tra i sistemi DAME e VOGCLUSTERS avviene attraverso lo scambio di documenti XML. Le componenti principali che vengono coinvolte nella comunicazione sono il *Framework*, dalla parte della suite DAME e il *Server* per la suite VOGCLUSTERS (Figura 6).

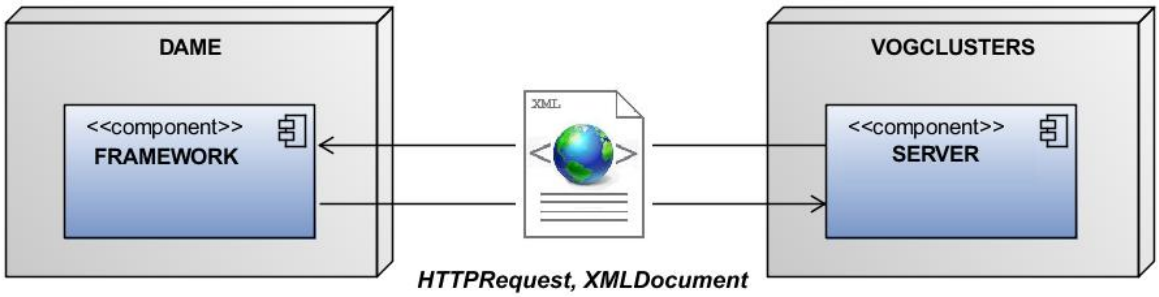

Figura 6: Comunicazione DAME - VOGCLUSTERS

Prima di analizzare i dettagli della comunicazione, è bene precisarne le principali motivazioni. La suite DAME, come anticipato nei paragrafi precedenti, ha un proprio database, differente dal database di VOGCLUSTERS. La comunicazione nasce dal bisogno di dover condividere le informazioni relative agli utenti registrati. Le due suite infatti, pur memorizzando i dati permanenti ognuna nel proprio database, hanno il vincolo di possedere una tabella comune. La tabella in questione è quella contenente le informazioni degli utenti registrati e risiede nel database di DAME. Questo spiega il perché dell'assenza di funzionalità come la registrazione e il login in VOGCLUSTERS. Tali funzionalità sono escluse perché ereditate dalla suite DAME.

Un tipico scenario che descrive la comunicazione tra i due sistemi è stato documentato nel seguente diagramma di sequenza.

Il caso d'uso analizzato è il Login dell'utente.

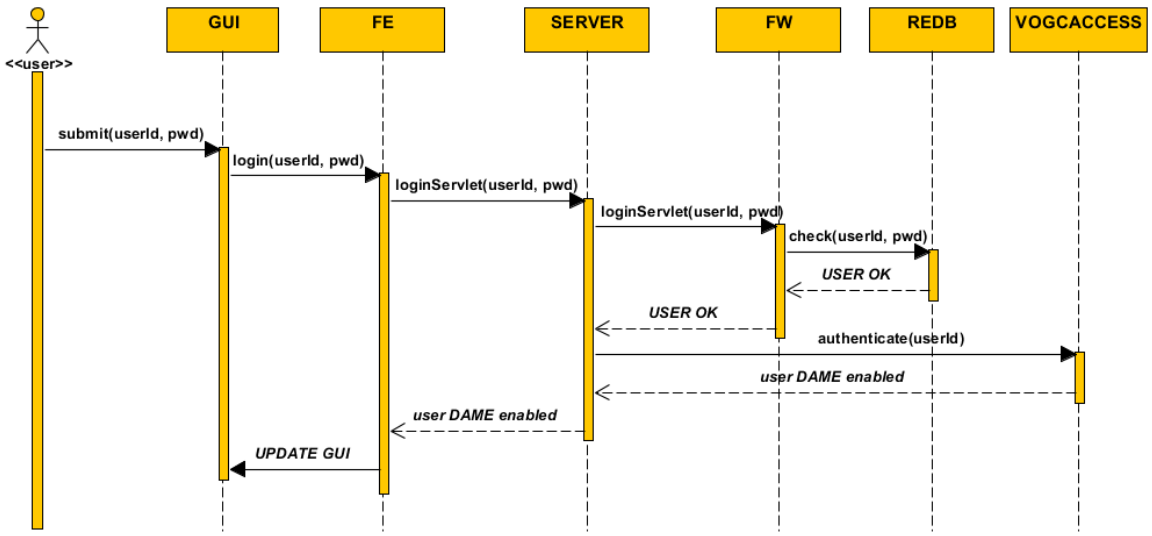

Figura 7: Sequence Diagram login utente

Come si intuisce dalla Figura 7, l'utente che esegue l'accesso a VOGCLUSTERS riceve il risultato dell'approvazione in modo invisibile, attraverso la comunicazione tra il suo componente Server ed il componente FW di DAME. In tal modo, se le credenziali dell'utente risultano pre-registrate nella base dati di DAME (component REDB), l'applicazione permetterà l'accesso, regolato sulla base del suo livello di autorizzazioni stabilito in DAME. Qualora l'utente non abbia precedentemente effettuato la registrazione in DAME, il sistema provvederà ad informarlo opportunamente.

Capitolo 2

Tecnologie utilizzate

2.1 SERVICE LAYER

La componente Server ha il compito principale di gestire la comunicazione tra la web application e le componenti di back-end del sistema (database, plotter). Dal punto di vista architetturale, tale componente è stata progettata seguendo lo stile REST (*Representational State Transfer*). Il termite REST è stato coniato nel 2000 da Roy Fielding, uno degli autori del protocollo HTTP, per rappresentare un sistema che permetta di descrivere ed identificare le risorse web. L'intero paradigma REST ruota intorno al concetto di risorsa. I servizi web infatti sono visti come risorse, ognuna delle quali è identificata da un *Uniform Resource Identifier* (URI).

Le applicazioni basate su REST dette anche *RESTful* usano richieste HTTP per inviare, leggere e cancellare dati. Quindi, tali applicazioni, sono in grado di manipolare le risorse attraverso l'utilizzo dei metodi GET, POST, PUT e DELETE del protocollo HTTP.

Il paradigma REST è stato visto come la soluzione ideale per la realizzazione della componente in questione. Infatti, ogni oggetto, quale ad esempio un ammasso globulare, un'immagine astronomica o un catalogo, viene identificato come una risorsa.

La componente è stata implementata utilizzando il linguaggio Java. Molteplici sono le motivazioni che hanno portato a tale scelta. Di seguito vengono indicate le principali.

- **Indipendenza dalla piattaforma**: Java può essere eseguito sulla maggior parte delle principali piattaforme hardware e software grazie al meccanismo della *Java Virtual Machine* che fa da filtro tra il codice Java e la macchina.

- **Sicurezza**: Progettato per creare software altamente affidabile, fornisce ampi controlli in fase di compilazione, seguito da ulteriori controlli in fase di esecuzione. Java permette di costruire applicazioni che possono difficilmente essere invase da altre applicazioni.

- **Semplicità**: il linguaggio Java è stato progettato per essere semplice da imparare e costruito sfruttando come base la sintassi e molte caratteristiche del C e del C++. Tuttavia per promuovere la sicurezza e la semplicità, ha tralasciato quegli elementi di C e C++ che conducevano alla complessità dei programmi e al verificarsi degli errori.

Uno dei modi per implementare un Web Service *RESTful* in Java è quello di realizzarlo attraverso l'utilizzo delle servlet.

2.1.1 LE SERVLET

Le servlet sono classi Java orientate alla comunicazione tra client e server. Una servlet riceve dei messaggi di richiesta da parte dei client e produce, in corrispondenza, dei messaggi di risposta. E' in esecuzione all'interno di un *Web Container*[4] secondo un particolare ciclo di vita.

Dal punto di vista dell'architettura, tutte le servlet devono implementare l'interfaccia `Servlet`. Questa interfaccia definisce cinque metodi:

1. `void init(ServletConfig config)`: questo metodo viene automaticamente chiamato una volta durante il ciclo di esecuzione di una servlet per inizializzarla. L'argomento `ServletConfig` viene fornito dal contenitore che esegue la servlet.

2. `ServletConfig getServletConfig()`: questo metodo restituisce un riferimento ad un oggetto che implementa l'interfaccia `ServletConfig`. Questo oggetto fornisce l'accesso alle informazioni di configurazione della servlet, come ad esempio i parametri di inizializzazione e il `ServletContext` della servlet, che fornisce alla servlet un accesso al suo ambiente (ossia il contenitore su cui è in esecuzione la servlet).

3. `String getServletInfo()`: questo metodo è definito dal programmatore di servlet per contenere le informazioni sulla servlet, come ad esempio l'autore e la versione.

4. `void service(ServletRequest request, ServletResponse response)`: questo è il metodo chiamato per rispondere alla richiesta di un client.

5. `destroy()`: questo metodo di "pulizia" viene chiamato quando una servlet viene terminata dal contenitore sul quale era in esecuzione. È un buon metodo per liberare una risorsa usata dalla servlet (per esempio un file aperto o una connessione di un database aperta).

Esistono due classi astratte che implementano l'interfaccia `Servlet`:

- `GenericServlet`: situata nel package `javax.servlet`, non orientata alla comunicazione con uno specifico protocollo.

[4] Un *Web Container* è l'ambiente di esecuzione per Servlet e JSP (Java Server Page). Ha il compito di gestire la comunicazione con i client, il ciclo di vita di una servlet, il multithreading e la sicurezza.

- `HttpServlet`: situata nel package `javax.servlet.http`, orientata alla comunicazione basata su protocollo HTTP.

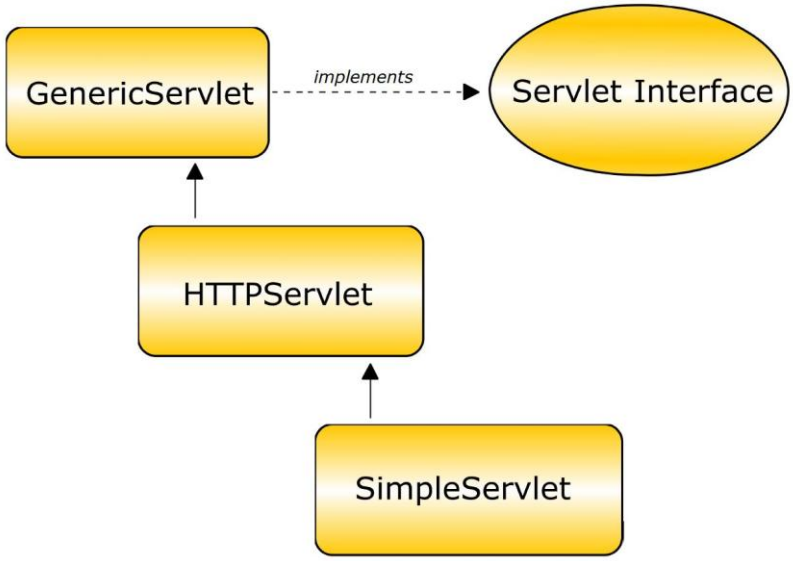

Figura 8: Classi Servlet

Una servlet Java è una classe che deriva dalla classe `GenericServlet`, presente nel package `javax.servlet`, o da una sua derivata. Le servlet realizzate nel progetto (rappresentate dal modulo *SimpleServlet* nella Figura 8) estendono tutte la classe `HttpServlet`. Questa classe deriva direttamente da `GenericServlet` ed è presente nel package `javax.servlet.http`.

Le classi di questo package estendono le funzionalità di base di una servlet supportando tutte le caratteristiche della trasmissione di dati con protocollo HTTP compresi cookies e richieste e risposte HTTP.

I metodi principali di una classe che estende `HttpServlet`, che coincidono con quelli maggiormente utilizzati nella fase implementativa del progetto, sono `doGet()` e `doPost()`. Altri metodi utilizzati meno di frequente sono `doPut()` e `doDelete()`. Tali metodi sono stati pensati per dare la possibilità di gestire in maniera differenziata le richieste HTTP in base al tipo (GET, POST, PUT o DELETE).

Oltre ad invocare uno di questi metodi vi è la possibilità di utilizzare un ulteriore metodo della classe `HttpServlet` denominato `service(HttpRequest, HttpResponse);` questa è dovuta al fatto che il *Servlet Container*, all'invocazione della servlet, chiama in primo luogo questo

metodo che a sua volta smista la richiesta al metodo opportuno (`doGet`, `doPost`, …). In ogni caso è sempre consigliabile implementare direttamente i metodi specifici.

L'oggetto di tipo `HttpServletRequest` passato come argomento dei metodi `doGet` e `doPost` è di fondamentale importanza perché permette di accedere ad una serie di informazioni che riguardano la richiesta HTTP giunta al server.

Un primo tipo di informazioni sono quelle relative alla "testata" della richiesta. Infatti la testata di un messaggio HTTP può contenere, sotto forma di coppie chiave-valore, una serie di informazioni, quali ad esempio il tipo di dispositivo che origina la richiesta, il browser utilizzato oppure altre informazioni utili a localizzare l'area geografica del client. Questi dati sono accessibili tramite i metodi `getHead(String headName)` o `getHeadNames()`.

Un secondo tipo di informazioni, di maggiore rilevanza rispetto a quelle appena descritte, sono i parametri della richiesta. Infatti, sia le richieste di tipo GET che quelle di tipo POST possono trasportare una serie di parametri che, anche in questo caso, saranno sotto forma di coppie chiave-valore. La differenza tra i due metodi consiste nella modalità con cui vengono codificati questi valori. Nel caso di GET tali valori sono riportati direttamente nell'URL (dando origine ad URL del tipo http://dominio.com/index?chiave1=valore1&chiave2=valore2), mentre nel caso di POST sono codificati all'interno del flusso di dati HTTP. In entrambi i casi, per accedere ai nomi dei parametri viene utilizzato il metodo `getParameterNames()`. Invece, conoscendo i nomi dei parametri è possibile accedere ai loro valori attraverso il metodo `getParameter(String nomeParametro)`.

Il secondo argomento dei metodi `doGet` e `doPost` è l'oggetto `HttpServletResponse`. Scopo principale di tale oggetto è quello di fornire la possibilità di inviare una risposta al client. Esistono due modalità: in primo luogo è possibile restituire un oggetto di tipo `javax.servlet.ServletOutputSream` che può essere utilizzato per inviare dati di tipo binario tramite i metodi `print` e `println`; in secondo luogo è possibile restituire l'oggetto `java.io.Printer` per inviare dati in formato testuale al client.

Una volta realizzata dallo sviluppatore, la servlet viene istanziata al momento opportuno dal *Web Container* che ne gestirà il ciclo di vita. Nella fase di inizializzazione, il *Web Container* invoca il metodo `init()` della servlet, viene creato un nuovo processo e la nuova servlet è pronta per ricevere e gestire le richieste provenienti dal client. Quando il *Web Container* riceve una nuova richiesta, crea un nuovo thread associato alla servlet e su di esso invoca il metodo `service()`, il quale a sua volta non fa altro che invocare il metodo `doGet()` oppure `doPost()` passandogli gli oggetti relativi alla *request* ed alla *response* (Figura 9).

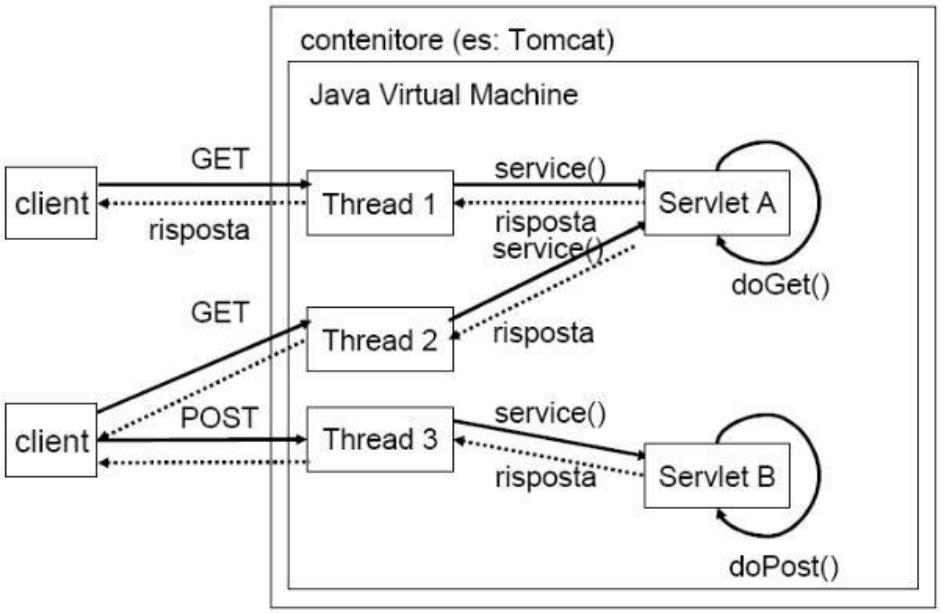

Figura 9: Esempio di servlet

La caratteristica principale di una servlet è che non viene creato un nuovo processo per ogni nuova richiesta, ma è in grado di gestire richieste concorrenti generando un thread per ognuna di essa. Proprio questa è la caratteristica che distingue una servlet da uno script CGI (*Common Gateway Interface*). Infatti quest'ultimo prevede l'allocazione di un nuovo processo CGI per ogni richiesta pervenuta al CGI server, appesantendo notevolmente il server man mano che gli accessi aumentano. Questo rende la Servlet notevolmente più veloce di un CGI.

2.2 COMUNICAZIONE SERVICE LAYER – FE LAYER

La comunicazione tra questi due strati avviene attraverso documenti XML. In un contesto del genere, risulta fondamentale la presenza di strumenti in grado di effettuare il parsing e la generazione di documenti XML. Infatti, il parsing è necessario quando una delle due componenti coinvolte nella comunicazione, riceve un documento XML. Il documento ricevuto, deve essere scansionato per poterne estrarne i contenuti. Invece, la generazione di documenti XML risulta necessaria quando la componente che deve inviare dati, deve inglobarli nel formato XML.

Nel caso specifico di VOGCLUSTERS, due strati del sistema necessitano di tali strumenti: il *Service Layer* e il *Frontend Layer*.

Nei successivi paragrafi verranno descritti dettagliatamente gli strumenti utilizzati sia per la generazione che per il parsing di documenti XML.

2.2.1 XML

Per lo scambio dei dati fra il server e il client si è scelto di utilizzare documenti in formato XML. L'XML è un formato di testo semplice e molto flessibile, derivato, come l'HTML (*HyperText Markup Language*), da SGML (*Standard General Markup Language*), uno standard internazionale (ISO 8879).

> *La "X" in XML sta per eXtensible: estendibilità significa che il linguaggio può essere ampliato per soddisfare esigenze specifiche. XML non è basato su un insieme finito di marcatori, e può essere esteso creando marcatori descrittivi adatti alle proprie necessità* (Shepherd s.d.).

L'XML è un meta-linguaggio per definire la struttura di documenti e dati. I documenti XML (come quelli HTML) sono formati da elementi o nodi. Ogni elemento XML è caratterizzato da un tag di apertura (come `<root>`), seguito da testo o altri tag e dal corrispondente tag di chiusura che inizia con `</` (nell'esempio `</root>`). Ecco un esempio di documento XML:

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

<root>
  <element1 attribute="value" >
    <subelement1 attribute="value" >data</subelement1>
  </element1>
  <element2>
    <subelement2 attribute="value" >data</subelement2>
  </element2>
</root>
```

Ad ogni etichetta possono essere associati degli attributi per aumentare il suo livello di dettaglio. Al contrario di un documento HTML, che viene interpretato ugualmente dal browser, anche se contiene errori di sintassi, un documento XML, per essere fruibile, deve essere sintatticamente corretto e ben formato, cioè deve possedere le seguenti caratteristiche:

- una riga di intestazione, che è la prima istruzione che appare scritta nel documento e ne specifica il tipo di versione e di codifica;
- un unico elemento radice (ovvero il nodo principale, `<root>` nell'esempio precedente) che contiene tutti gli altri nodi del documento;
- all'interno del documento tutti i tag devono essere chiusi correttamente;
- XML fa distinzione tra maiuscole e minuscole, per cui i nomi dei tag e degli attributi devono coincidere nei tag di apertura e chiusura anche in relazione a questo aspetto;

Oltre ad essere ben formato, un documento XML deve essere valido. Un documento per essere valido deve sottostare ad una serie di regole. In altre parole, bisogna definire una "grammatica" per ogni documento XML ideato. In generale, una grammatica è un insieme di regole che indica quali vocaboli (elementi) possono essere utilizzati e con che struttura è possibile comporre frasi (documenti).

Attualmente esistono due modi per definire grammatiche per documenti XML: DTD (*Document Type Definition*) ed *XML Schema*.

Un DTD è un documento che descrive i tag utilizzabili in un documento XML, la loro reciproca relazione nei confronti della struttura del documento e altre informazioni sugli attributi di ciascun tag. È possibile trovare le regole formali per le DTD sul sito ufficiale[5] del W3C (*World Wide Web Consortium*).

DTD è il più vecchio formato di schema definito. Per questa ragione esso è universalmente supportato, ma ha anche alcuni limiti: una certa rigidità nella definizione del documento e la mancanza di supporto per le innovazioni delle versioni successive di XML, come i *namespaces*, che consentono di risolvere ambiguità nell'assegnazione dei nomi agli elementi, separandoli in diversi domini. Un vantaggio di DTD è però la sua semplicità.

Uno *XML Schema* è un documento XML che utilizza un insieme di tag speciali per definire la struttura di un documento XML. Da questa definizione emerge subito una novità interessante: viene utilizzato un documento XML per validarne un altro. Il suo vantaggio principale è quello di essere più versatile, consentendo di imporre ai documenti vincoli più complessi. Ad esempio si possono imporre vincoli sui tipi di dati del documento, cosa impossibile in DTD. Il vincolo che si deve rispettare nella creazione di uno schema è l'uso dei tag definiti dall'*XML Schema Language* specificati dal W3C nei documenti ufficiali[6]. Lo schema adottato in VOGCLUSTERS è di tipo *XML Schema*, questo perché si è preferita la sua maggiore flessibilità rispetto a DTD. Lo schema finale utilizzato è denominato VOTable.

2.2.2 SCHEMA VOTABLE

Lo schema VOTable è uno standard XML realizzato per rappresentare dati (storicamente sorto all'interno della Collaborazione IVOA per il trattamento standardizzato di dati astronomici) in forma tabulare. Quando si parla di dati astronomici, una tabella può essere vista come un insieme

[5] http://www.w3.org/TR/REC-xml/
[6] http://www.w3.org/TR/xmlschema-0/
http://www.w3.org/TR/xmlschema-1/
http://www.w3.org/TR/xmlschema-2/

non ordinato di righe, ognuna delle quali contenente dati di un formato specifico. Ogni riga della tabella è una sequenza di celle, e ognuna di esse può contenere un tipo di dato primitivo oppure un array di dati primitivi. È possibile riassumere i principali aspetti che caratterizzano una VOTable nei punti che seguono:

- **Separazione fra metadati e dati**: i metadati, cioè i nomi delle colonne, vengono definiti in una sezione separata dai loro valori effettivi. VOTable pone una grossa enfasi sui metadati, permettendo di organizzare le colonne in gruppi e di specificarle in maniera precisa, definendo parametri come il loro tipo, la loro grandezza e la precisione dei loro valori. Questa separazione inoltre può consentire un *parsing* più efficiente di un documento. Infatti se è necessario conoscere solo la struttura di una tabella, non bisognerà scorrere l'intero documento, ma basterà fermarsi alla sezione di definizione dei metadati.

- **Utilizzo degli UCD (*Unified Content Descriptors*)**: ad ogni colonna di una tabella viene associato un UCD. Questi sono particolari stringhe utilizzate nel VO per specificare la natura semantica di un certo dato (definizione di ontologie) e per discriminarne univocamente la categoria di appartenenza in ambito astrofisico. Gli UCD descrivono infatti quantità astronomiche e sono costruiti combinando termini specifici provenienti da un vocabolario (in continua evoluzione) definito dal VO[7]. Un UCD quindi, non definisce il nome di una colonna, ma rappresenta un identificatore unico che specifica la categoria ontologica della quantità contenuta. Ad esempio, `phys.temperature` rappresenta una temperatura senza specificarne la particolare unità. La natura gerarchica degli UCD, comunque ci permette di definire l'ucd di secondo livello `phys.temperature.effective` in grado di fornire una maggiore informazione sul tipo di temperatura descritto. Un client può quindi effettuare il *parsing* di una VOTable senza sapere il nome o la posizione effettiva di una colonna, ma limitandosi a considerare gli UCD a cui è interessato.

  Nella Figura 10 viene mostrata l'utilità e l'importanza degli UCD. Come si può notare viene fornito l'esempio di due servizi web che fanno riferimento alle stesse quantità astronomiche, ma le nominano in maniera differente. Grazie alla presenza degli UCD, è possibile determinare univocamente di che quantità si tratta.

[7] http://www.ivoa.net/Documents/latest/UCDlist.html

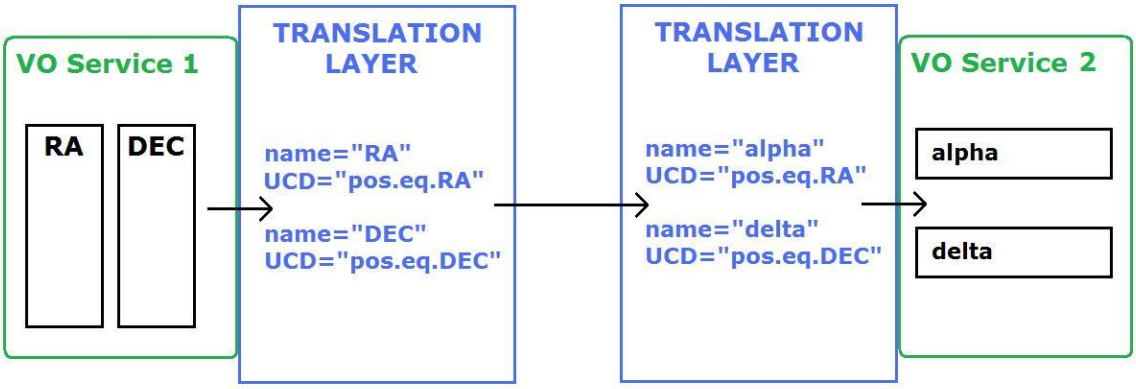

Figura 10: Utilizzo degli UCD

- **Struttura ricorsiva**: alcuni elementi possono avere al loro interno sotto elementi del loro stesso tipo, in modo da rappresentare elementi più complessi.

All'inizio di ogni schema VOTable, dopo un *header* contenente informazioni sulla versione di XML utilizzata, viene posto l'elemento VOTABLE. Tra i sottoelementi di VOTABLE ci sono, solitamente, elementi come DESCRIPTION e INFO, che permettono di definire parametri generici, riguardanti il documento. Gli elementi più importanti contenuti in una VOTABLE sono però le RESOURCEs. Una risorsa è un insieme di tabelle correlate. Oltre ad alcuni parametri descrittivi della risorsa, questa contiene uno o più elementi TABLE, rappresentanti le tabelle del documento. Tra gli elementi più importanti di una tabella ci sono i campi FIELD e PARAM. Un campo FIELD rappresenta una colonna della tabella. Un PARAM è invece una costante. Questi campi possono avere vari attributi, come un UCD che definisca meglio il tipo del campo. Ogni tabella contiene una sezione TABLEDATA, dove sono inseriti i dati delle colonne corrispondenti.

Per comprendere meglio la struttura, di seguito è stato riportata una VOTable di esempio.

```
<?xml version="1.0"?>
<VOTABLE version="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ivoa.net/xml/VOTable/VOTable/v1.1">
   <RESOURCE name="myFavouriteGalaxies">
     <TABLE name="results">
       <DESCRIPTION>Velocities and Distance estimations</DESCRIPTION>
       <PARAM name="Telescope" datatype="float" ucd="phys.size;instr.tel" unit="m"
              value="3.6"/>
       <FIELD name="RA" ID="col1" ucd="pos.eq.ra;meta.main" ref="J2000"
              datatype="float" width="6" precision="2" unit="deg"/>
       <FIELD name="Dec" ID="col2" ucd="pos.eq.dec;meta.main" ref="J2000"
              datatype="float" width="6" precision="2" unit="deg"/>
       <FIELD name="Name" ID="col3" ucd="meta.id;meta.main"
              datatype="char" arraysize="8*"/>
       <FIELD name="RVel" ID="col4" ucd="src.veloc.hc" datatype="int"
              width="5" unit="km/s"/>
       <FIELD name="e_RVel" ID="col5" ucd="stat.error;src.veloc.hc"
              datatype="int" width="3" unit="km/s"/>
        <FIELD name="R" ID="col6" ucd="phys.distance" datatype="float"
               width="4" precision="1" unit="Mpc">
           <DESCRIPTION>Distance of Galaxy, assuming H=75km/s/Mpc</DESCRIPTION>
        </FIELD>
```

```
        <DATA>
           <TABLEDATA>
              <TR>
                <TD>010.68</TD>
                <TD>+41.27</TD>
                <TD>N 224</TD>
                <TD>-297</TD>
                <TD>5</TD>
                <TD>0.7</TD>
              </TR>
              <TR>
                <TD>287.43</TD>
                <TD>-63.85</TD>
                <TD>N 6744</TD>
                <TD>839</TD>
                <TD>6</TD>
                <TD>10.4</TD>
              </TR>
              <TR>
                <TD>023.48</TD>
                <TD>+30.66</TD>
                <TD>N 598</TD>
                <TD>-182</TD>
                <TD>3</TD>
                <TD>0.7</TD>
              </TR>
           </TABLEDATA>
        </DATA>
      </TABLE>
   </RESOURCE>
</VOTABLE>
```

Nell'esempio è presente l'elemento root VOTABLE con un singolo elemento RESOURCE composto da un'unica tabella di 6 colonne. Ogni colonna è descritta da un elemento FIELD. Oltre agli elementi FIELD, è presente anche un elemento addizionale PARAM. I valori corrispondenti ai FIELD descritti, sono contenuti nell'elemento DATA, in particolare in TABLEDATA. Qui devono essere sempre presenti tanti elementi TD quanti sono gli elementi FIELD descritti in precedenza. Gli elementi TD sono racchiusi negli elementi TR e sono quelli che contengono i dati effettivi. Si può pensare ad un elemento TR come all'inizio di una tupla di una tabella. In ogni tupla saranno presenti le colonne (scandite dagli elementi TD) con i rispettivi dati.

2.2.3 GENERATORE DI XML

Per semplificare la creazione dei documenti XML, abbiamo deciso di associare un *template* ad ogni documento. Con il termine *template* si intende un modello di documento nel quale possono essere importati dati dell'esterno. Esso può essere visto come un modello generico rappresentante la struttura di un qualunque documento.

Definendo i template secondo i vincoli dello schema VOTable, è stato possibile quindi creare, a partire da questi, documenti XML diversi (di contenuto, ma con il medesimo schema) ben formati e validi. Come strumento di sviluppo per la generazione di tali documenti è stato utilizzato FreeMarker.

2.2.3.1 FREEMARKER

FreeMarker[8] è un strumento che consente di generare testo dinamicamente a partire da un *template*. FreeMarker non è un'applicazione per gli utenti finali, ma consiste in una serie di librerie scritte in Java utilizzabili dai programmatori per sviluppare le proprie applicazioni (Figura 11).

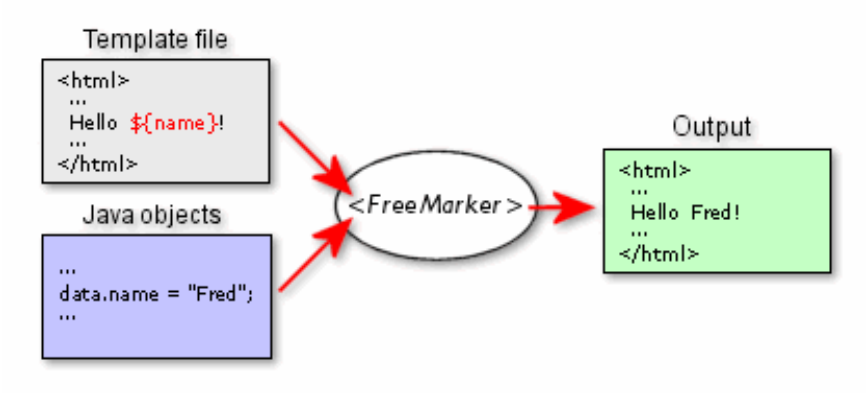

Figura 11: Schema Freemarker

Come si può notare dalla Figura 11, nel *template* (modulo *Template file*) la porzione di testo che dovrà essere generata dinamicamente da FreeMarker è contrassegnata con il simbolo ${ }. Un'applicazione che utilizza FreeMarker, preleva le informazioni dagli oggetti Java per poi passarle al motore di FreeMarker. Quest'ultimo genererà un documento XML a *runtime* sostituendo la variabile presente tra le parentesi graffe ${ } con le informazioni prelevate dall'oggetto.

Ogni *template* può utilizzare diverse variabili da riempire con i dati reali durante la fase di generazione. L'insieme delle variabili di un *template* viene chiamato data-model. Il processo di generazione viene effettuato tramite delle API Java. Un programmatore deve semplicemente caricare il *template* da lui richiesto e definire un proprio *data-model*, contenente i valori effettivi delle variabili del modello.

I *template* sono dei programmi scritti in un linguaggio chiamato FTL (*FreeMarker Template Language*). Un *template* (cioè un programma FTL) presenta quattro elementi principali:

- **Testo**: il testo è l'unica cosa che verrà riportata sul documento generato senza subire variazioni.
- **Interpolazione**: le interpolazioni sono delimitate dal simbolo ${ e }. Il testo presente all'interno verrà sostituito con il valore della variabile.
- **Etichette FTL**: le etichette FTL sono delle particolari istruzioni chiamate direttive che vengono interpretate da Freemarker ma non vengono riportare nel documento generato.

[8] http://freemarker.org/

- **Commenti**: i commenti sono delimitati dai simboli <#-- e -->. I commenti vengono ignorati da Freemarker e non saranno riportati nel documento generato.

Nei *template* realizzati le uniche direttive utilizzate sono #list per scorrere una lista di elementi, #if per gestire le condizioni e #assign per assegnare un valore ad una variabile locale. Per una migliore comprensione, di seguito viene mostrato uno dei *template* realizzati. In rosso sono state poste le interpolazioni e in blu le direttive.

```
<?xml version="1.0" ?>
<VOTABLE version="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1
http://www.ivoa.net/xml/VOTable/v1.1"
xmlns="http://www.ivoa.net/xml/VOTable/v1.1">
   <DESCRIPTION>Users DAME abilitated Informations</DESCRIPTION>
   <RESOURCE>
      <TABLE name='Users Informations'>
         <FIELD name='UserId' datatype='char' />
         <FIELD name='Name' datatype='char' />
         <FIELD name='Surname' datatype='char' />
         <FIELD name='Motivation' datatype='char' />
         <FIELD name='Status' datatype='char' />
         <DATA>
            <TABLEDATA>
               <#list userInfo as info>
               <TR>
                  <TD>${info.userId}</TD>
                  <TD>${info.name}</TD>
                  <TD>${info.surname}</TD>
                  <TD>${info.motivation}</TD>
                  <TD>${info.active}</TD>
               </TR>
               </#list>
            </TABLEDATA>
         </DATA>
      </TABLE>
   </RESOURCE>
</VOTABLE>
```

2.2.4 *PARSER* DI XML

Per elaborare un documento XML è necessario eseguire il *parsing* (ovvero l'analisi) del documento. Un *parser* è un programma che legge un file, conferma che abbia il formato corretto e ricostruisce la struttura del documento, in modo da permettere al programmatore di accedere agli elementi che lo costituiscono.

La libreria Java mette a disposizione due generi di *parser* XML:

- **DOM (*Document Object Model*)**: legge un documento XML in una struttura ad albero;

- **SAX (*Simple API for XML*)**: genera eventi mentre legge un documento XML;

Il *parser* DOM legge l'intero documento XML in una struttura dati ad albero. Nella maggior parte dei casi DOM è più semplice da utilizzare e funziona senza problemi. Conviene tuttavia prendere in esame SAX quando si devono elaborare documenti lunghi, nei quali una struttura ad albero occuperebbe troppa memoria, oppure quando si è interessati ad alcuni elementi indipendentemente dal loro contesto.

2.2.4.1 JDOM

Dopo un'attenta analisi delle nostre esigenze, la scelta è caduta su JDOM[9]. JDOM è un API *open source* che tenta di risolvere alcune insufficienze riconosciute dai *parser* SAX e DOM fornendo, nella maggior parte dei casi, un'alternativa a questi. L'idea di base di JDOM è quella di fornire uno strumento per la gestione dei documenti XML con i meccanismi che si aspetterebbe uno sviluppatore Java.

JDOM non è basata sul DOM o su SAX, ma piuttosto permette ad un utente di gestire un documento XML in una struttura ad albero evitando tutte le particolarità del DOM. Allo stesso tempo, garantisce un livello di prestazioni pari a quello offerto da SAX, permettendo di effettuare un'analisi e ottenere un output molto velocemente. In aggiunta, supporta i *namespace* e la convalida attraverso i DTD e *XML Schema*.

I programmi sviluppati con JDOM possono ricevere in input documenti XML da un *parser* SAX/DOM e analogamente possono generare un documento DOM o uno *stream* di eventi SAX.

Un documento XML viene rappresentato come un'istanza della classe *org.jdom.Document*. Non restituisce oggetti sotto forma di *NodeList* o di *Attributes*, ma classi di collezione Java 2, come `List` e `Map`, aumentando così il grado di semplicità. La prima caratteristica Java-oriented di JDOM che lo sviluppatore incontra è quella di poter creare gli oggetti direttamente con il proprio costruttore, senza la necessità di utilizzare metodi *factory*. Per creare ad esempio un oggetto org.jdom.Element che rappresenta un tag `<Document>` è quindi sufficiente scrivere:

```
Element e = new Element("Document");
```

Per il *parsing* di documenti XML, JDOM fornisce i *builder*, oggetti che costruiscono un documento a partire da diverse sorgenti dati come file e InputStream. Sono definiti due tipi di *builder*: DOMBuilder e SAXBuilder. Come è evidente dal nome, DOMBuilder carica un documento a partire da un oggetto org.w3c.dom.Document mentre SAXBuilder sfrutta un *parser* SAX ed è

[9] http://www.jdom.org/

quindi più performante. Il seguente codice mostra come viene utilizzato SAXBuilder per eseguire il *parsing* di un file istanziando un documento JDOM:

```
SAXBuilder builder = new SAXBuilder(true);

Document doc = builder.build(new File(arg[0]));
```

Il modello ad oggetti di JDOM permette di navigare nella struttura di un documento in maniera molto semplice. Ad esempio, dato un *Document*, è possibile accedere all'elemento radice con:

```
Element root = doc.getRootElement();
```

A questo punto si può accedere alla lista di tutti i nodi figli

```
List childrenList = root.getChildren();
```

o a tutti i nodi figli aventi lo stesso nome

```
List childrenList = root.getChildren("CHAPTER");
```

Le precedenti linee di codice sono molto semplici e allo stesso tempo dimostrano quanto JDOM sia un strumento vicino allo sviluppatore Java: il metodo `getChildren()` restituisce un oggetto `java.util.List` che lo sviluppatore può utilizzare applicando i metodi a cui è abituato. Per accedere ad un singolo nodo figlio si usa il metodo `getChild()`. In questo caso si accede al primo nodo CHAPTER partendo dalla radice del documento

```
Element e = root.getChild("CHAPTER");
```

Per accedere al contenuto di un elemento si usa il metodo `getContent()`

```
String value = e.getContent();
```

Nel caso in cui l'elemento abbia un attributo lo si può ottenere come stringa

```
String attValue = e.getAttributeValue("name");
```

In conclusione JDOM è uno strumento veramente interessante per i seguenti motivi:

- Il suo modello ad oggetti è molto più pratico di DOM. Ad esempio poter creare oggetti direttamente con il costruttore è sicuramente più immediato che usare metodi *factory*;
- L'utilizzo delle *collection* semplifica inoltre notevolmente il codice necessario ed è una caratteristica di programmazione molto vicina al programmatore Java.

Il futuro di JDOM è molto promettente anche perché il progetto è stato proposto come *Java Specification Request* e potrebbe essere incluso in una delle prossime versioni di Java.

2.3 FRONTEND LAYER

La continua evoluzione di internet negli ultimi anni ha alimentato la diffusione di applicazioni web sempre più evolute. Le *web application* sono divenute popolari, grazie alla possibilità offerta ai client di accedere all'applicazione utilizzando normali *web browser*. Questo tipo di applicazioni, nel gergo informatico, vengono denominate RIA (*Rich Internet Application*).

Nel modello tradizionale, l'elaborazione delle informazioni richieste e la conseguente creazione di pagine web sono tutte a carico del server. Il dinamismo delle pagine web è generato unicamente dal server, mentre il client si limita ad interpretare i dati che gli vengono inviati, senza elaborarli.

Per questo motivo, con l'andare del tempo, si è pensato di introdurre un dinamismo analogo lato client, direttamente sul browser dell'utente.

Le RIA sono applicazioni interattive e veloci, in quanto la parte di elaborazione dei dati è inserita nel browser del client, mentre i dati risiedono sul server. In particolare queste applicazioni possiedono le caratteristiche e le funzionalità delle tradizionali applicazioni per computer, senza però necessitare dell'installazione locale sul disco fisso.

Le RIA si caratterizzano per la dimensione interattiva e per la velocità d'esecuzione. Infatti la parte dell'applicazione che elabora i dati è trasferita a livello client e fornisce una pronta risposta all'interfaccia utente, mentre la gran parte dei dati e dell'applicazione rimane sul server remoto, con notevole alleggerimento per il computer utente.

Nei paragrafi che seguiranno verrà affrontato lo studio delle principali tecnologie per lo sviluppo di RIA.

2.3.1 AJAX

AJAX (*Asynchronous JavaScript*[10] *And XML*) è una tecnica di sviluppo per creare applicazioni web interattive. L'intento di tale tecnica è quello di ottenere pagine che rispondono in maniera più rapida, grazie allo scambio in background di dati con il server, in modo che l'intera pagina web non debba essere ricaricata ogni volta che necessiti di una modifica.

[10] JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web. Fu originariamente sviluppato da Brendan Eich della Netscape Communications con il nome di Mocha e successivamente di LiveScript, ma in seguito è stato rinominato "JavaScript" ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Sun Microsystems.

Il concetto è in parte espresso nell'acronimo scelto, un utilizzo asincrono di Javascript che, attraverso l'interfacciamento con XML, può permettere ad un client di richiamare informazioni lato server in modo veloce e trasparente, allargando gli orizzonti delle RIA.

Come si può evincere dall'acronimo, AJAX non è una nuova tecnologia, ma un insieme di diverse tecnologie già affermate, che cooperano tra di loro. In particolare un'applicazione AJAX è costituita da:

- Una rappresentazione di base con XHTML[11] e CSS[12];
- Un'interfaccia dinamica grazie all'interazione di JavaScript con il DOM;
- Uno scambio e manipolazione di dati tramite XML;
- Un reperimento asincrono dei dati tramite l'oggetto XMLHttpRequest;

Ma analizziamo in dettaglio quali sono le principali differenze con il modello tradizionale. In un'applicazione web tradizionale, ogni azione dell'utente sull'interfaccia grafica corrisponde ad una richiesta HTTP al server. Il server ogni volta che riceve una richiesta, elabora i dati (comunicando eventualmente con un database) e restituisce in risposta al client una pagina HTML.

È facile comprendere che un simile approccio non è l'ideale per le moderne applicazioni web. Infatti l'interazione dell'utente si arresta ogni volta che l'applicazione ha bisogno di qualcosa dal server e inoltre, per fare un aggiornamento anche minimo sulla pagina, questa deve essere ricaricata completamente, con spreco di memoria sul server, di banda e di tempo.

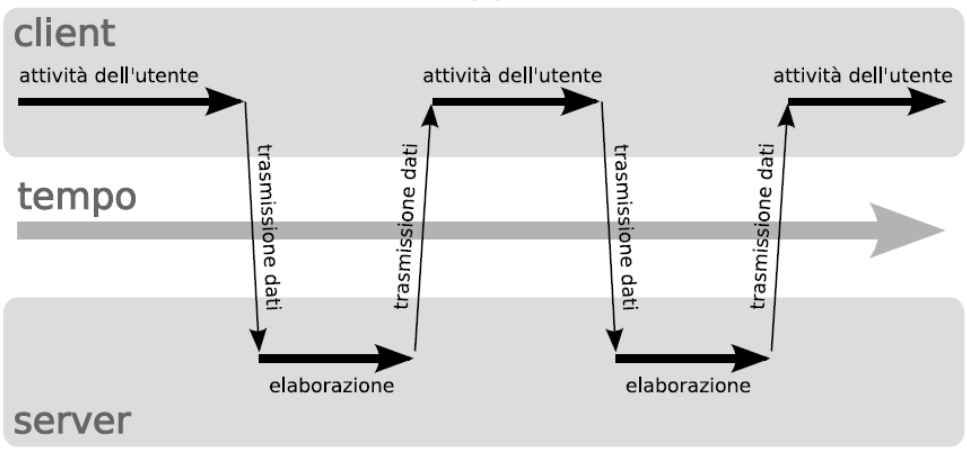

Figura 12: modello di richiesta sincrona tradizionale

[11] *eXtensible HyperText Markup Language* è un linguaggio di marcatura che associa alcune proprietà dell'XML con le caratteristiche dell'HTML: un file XHTML è un pagina HTML scritta in conformità con lo standard XML.
[12] I *Cascading Style Sheet* o detti anche semplicemente fogli di stile, vengono usati per definire la rappresentazione di documenti HTML, XHTML e XML.

Con AJAX la situazione è differente. La richiesta è asincrona, il che significa che non bisogna necessariamente attendere che sia stata ultimata per effettuare altre operazioni, stravolgendo sotto diversi punti di vista il flusso dati tipico di una pagina web. Mentre l'utente è all'interno della stessa pagina le richieste sul server possono essere numerose e completamente indipendenti.

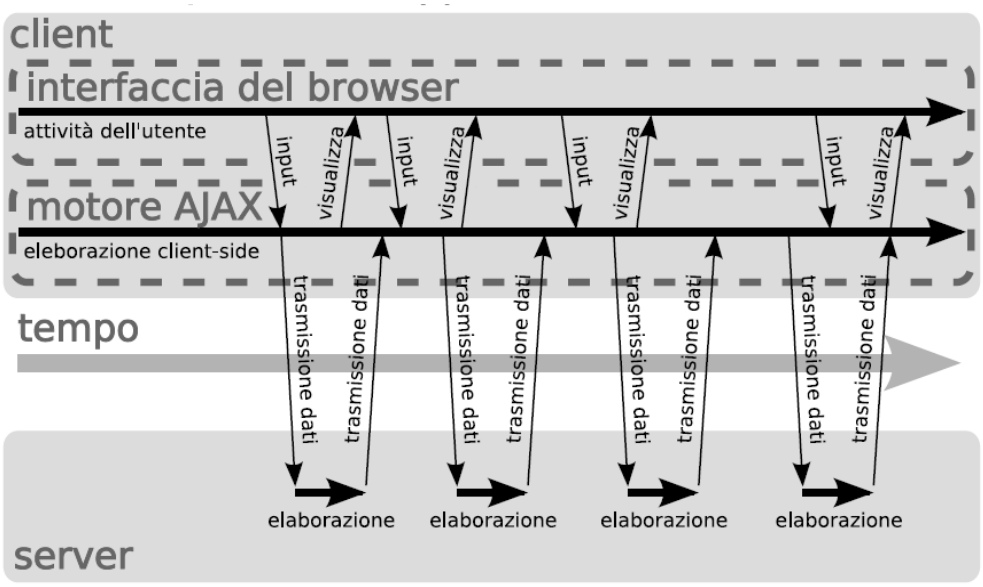

Figura 13: modello di richiesta asincrona

Un'applicazione AJAX elimina la discontinuità dell'interazione con il web, introducendo un intermediario tra l'utente ed il server: il motore AJAX. Di seguito verranno elencati i passi principali di una richiesta AJAX . Analizziamo i punti fondamentali di una richiesta AJAX nella figura che segue:

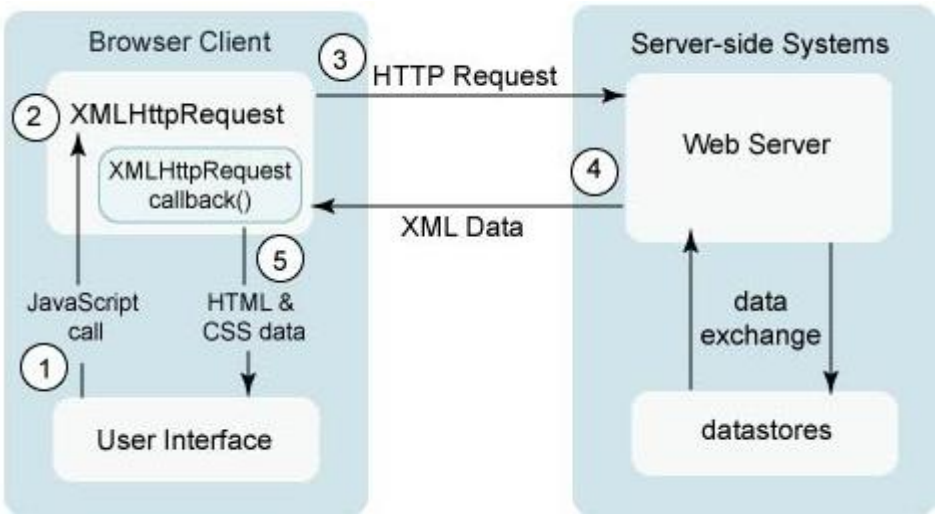

Figura 14: Sequenza di una richiesta AJAX

1. L'utente genera un evento, come il click di un bottone. Un evento del genere equivale ad una chiamata Javascript;

2. Viene creato un oggetto XMLHttpRequest e viene configurato con i parametri della richiesta che sarebbero l'ID del componente che ha generato l'evento e qualsiasi valore inserito dall'utente;

3. L'oggetto XMLHttpRequest effettua una chiamata asincrona al server. Quest'ultimo riceve la richiesta, la processa ed eventualmente memorizza dati nel database.

4. L'oggetto che ha processato la richiesta lato server, restituisce un documento XML contenente gli aggiornamenti richiesti dal client.

5. L'oggetto XMLHttpRequest riceve il documento XML e lo processa aggiornando la pagina HTML con i nuovi dati.

Tutte le chiamate in background sono opera del vero nucleo di AJAX: l'oggetto XMLHttpRequest. A seconda del browser usato prende nomi differenti o viene richiamato in maniera differente. Nel caso di Internet Explorer, ad esempio, questo oggetto si chiama XMLHTTP ed è restituito da un ActiveXObject mentre nei browsers alternativi più diffusi (Mozilla, Safari, FireFox, Netscape, Opera ed altri) XMLHttpRequest è supportato nativamente.

Come affermato in precedenza, la novità di AJAX sta sostanzialmente nel nuovo utilizzo di tecnologie già affermate da anni. Pur essendo una tecnologia molto innovativa, ha impiegato diversi anni per diffondersi ed affermarsi. Uno dei problemi principali infatti sta nella comprensione da parte degli sviluppatori su come utilizzare questo nuovo approccio in maniera efficiente. Ed è proprio questo uno dei problemi che tutt'oggi continua ad esistere. AJAX non si può di certo affermare che sia una tecnologia semplice da utilizzare, anzi tutt'altro.

Sviluppare una web application con un tecnologia AJAX implica una massiccia conoscenza di JavaScript. Ed è cosa nota che rendere il codice JavaScript efficiente e privo di bug è un compito piuttosto impegnativo.

L'aumento di complessità fa aumentare anche la probabilità di errore e può rallentare la produttività degli sviluppatori. Inoltre bisogna sempre tener conto della compatibilità cross-browser. Fortunatamente esistono alcuni framework che ci permettono di aggirare molti di questi problemi. Ma nessun strumento risulta pienamente soddisfacente. Solo negli ultimi anni è stato introdotto dalla Google il strumento di sviluppo GWT[13] (*Google Web Toolkit*).

[13] http://code.google.com/intl/it-IT/webtoolkit/

2.3.2 GWT

GWT è un framework open source, sviluppato da Google, che permette di creare applicazioni Web con AJAX utilizzando il linguaggio Java.

Ciò senza la necessità di scrivere codice JavaScript, HTML o CSS, a meno che non sia assolutamente necessario. Utilizzando l'API GWT, è possibile creare intere applicazioni Java. Il compilatore GWT traduce tutto il codice Java in codice del browser (HTML, CSS e JavaScript), incapsula l'oggetto XMLHttpRequest e minimizza i problemi di incompatibilità con i diversi browser.

Questo tipo di framework rende lo sviluppo di applicazioni web simile allo sviluppo di applicazioni tradizionali: tutto il codice viene scritto dallo sviluppatore e poi compilato per l'esecuzione. Nel caso di GWT, la compilazione comprende la creazione dei file HTML e JavaScript che consentiranno l'esecuzione dell'applicazione in un browser.

GWT si propone di emulare lo stile di programmazione Swing[14], fornendo strumenti che consentano di combinare facilmente diversi *widget*[15] nell'interfaccia utente. Avvalendosi così della vasta conoscenza del linguaggio Java, posseduta dalla gran parte degli sviluppatori, questo nuovo framework si propone di migliorare la produttività e ridurre i tradizionali tempi di sviluppo di una web application basata su AJAX.

Le applicazioni sviluppate mediante GWT possono essere eseguite in due modalità differenti: *hosted mode* e *web mode*.

Mediante la modalità *hosted mode*, l'applicazione viene lanciata come *bytecode* Java all'interno della Java Virtual Machine. In questa modalità, sono utilizzabili tutti gli strumenti di debug disponibili nell'ambiente di sviluppo. Come supporto alla modalità *hosted mode*, GWT fornisce uno speciale browser[16] (in realtà non è realmente un browser, ma una sorta di versione emulata del browser di default).

Mediante la modalità *web mode*, invece, l'applicazione viene eseguita come una classica applicazione Web. Il compilatore GWT (analizzato di seguito) genera il codice HTML e JavaScript, interpretabile dal browser Web.

[14] Swing è un framework per Java orientato allo sviluppo di interfacce grafiche. Parte delle classi del framework Swing sono implementazioni di widget come caselle di testo, pulsanti, pannelli e tabelle.
[15] Nell'ambito della programmazione, un widget è un elemento (tipicamente grafico) dell'interfaccia utente di un programma che facilita all'utente l'interazione con il programma stesso.
[16] Un browser o navigatore è un programma che consente di visualizzare i contenuti delle pagine dei siti web e di interagire con essi, permettendo così all'utente di navigare in internet. Il browser è infatti in grado di interpretare l'HTML

L’architettura di GWT è costituita da quatto componenti fondamentali (Figura 15).

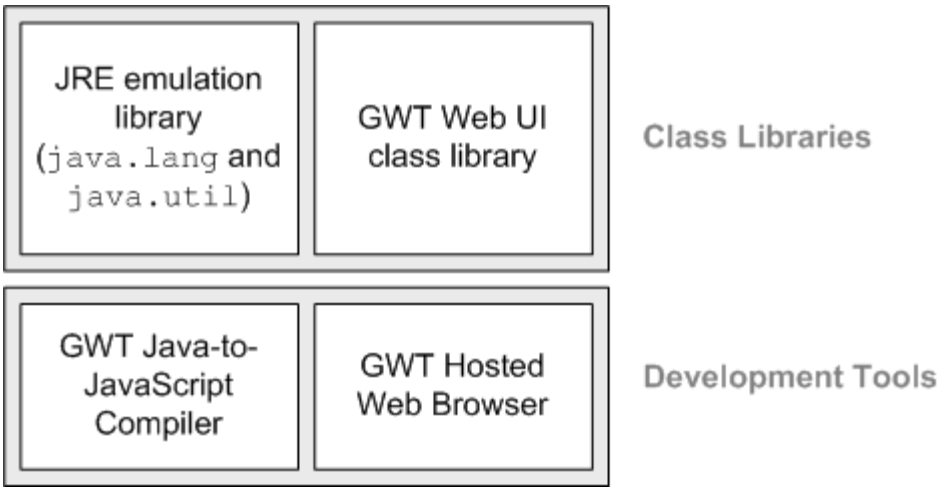

Figura 15: Architettura GWT

Analizziamo singolarmente ogni modulo dell’architettura:

- **GWT Java-to-JavaScript Compiler**: viene considerato il cuore di GWT. Questo è ciò che rende GWT un potente strumento per la costruzione di RIA. Il compilatore fa in modo che tutto il codice Java scritto venga poi tradotto in Javascript (anche per versioni specifiche del browser). Deve essere utilizzato quando si vuole eseguire l'applicazione in modalità *web mode*.

- **JRE Emulation Library**: prima di passare in *web mode*, GWT deve assicurarsi che il codice Java non utilizzi classi “proibite”, ovvero che non siano traducibili in JavaScript. In *hosted mode*, GWT usa la JRE *Emulation Library* per verificare la validità del nostro codice. Questa libreria contiene le implementazioni in linguaggio Javascript delle librerie Java standard maggiormente utilizzate (package java.lang.*, java.util.*).

- **UI Building Library**: questa libreria fornisce un insieme di interfacce personalizzate e di classi che consentono di creare widget vari, come pulsanti, caselle di testo, immagini e testo, utili alla costruzione dell’interfaccia grafica della web application.

- **GWT Hosted Web Browser**: permette di eseguire le applicazioni in *hosted mode*. Il codice Java è eseguito nella JVM senza convertirlo in JavaScript/HTML.

Dopo aver analizzato quali siano le componenti fondamentali del framework GWT possiamo procedere con l’analisi di un altro aspetto fondamentale: la comunicazione client-server in GWT.

GWT si avvale del meccanismo RPC (*Remote Procedure Call*) e della serializzazione[17] degli oggetti per lo scambio di messaggi fra il client e il server. Una RPC è una funzione implementata lato server e invocata da un client. La comunicazione consiste nell'invio e nella ricezione di oggetti serializzati. In linee generali il server espone delle risorse (o servizi) che vengono poi richiamate in maniera asincrona dal client. Se infatti il client vuole invocare un qualsiasi metodo del server, deve creare un apposito oggetto *AsyncCallback* in grado di poter gestire la risposta che arriverà in un momento successivo. L'idea fondamentale è fornire la capacità di sfruttare i metodi di una classe remota, realizzata su di un server, effettuando una chiamata locale lato client.

Di seguito viene analizzato nel dettaglio quali sono le componenti coinvolte in questo meccanismo.

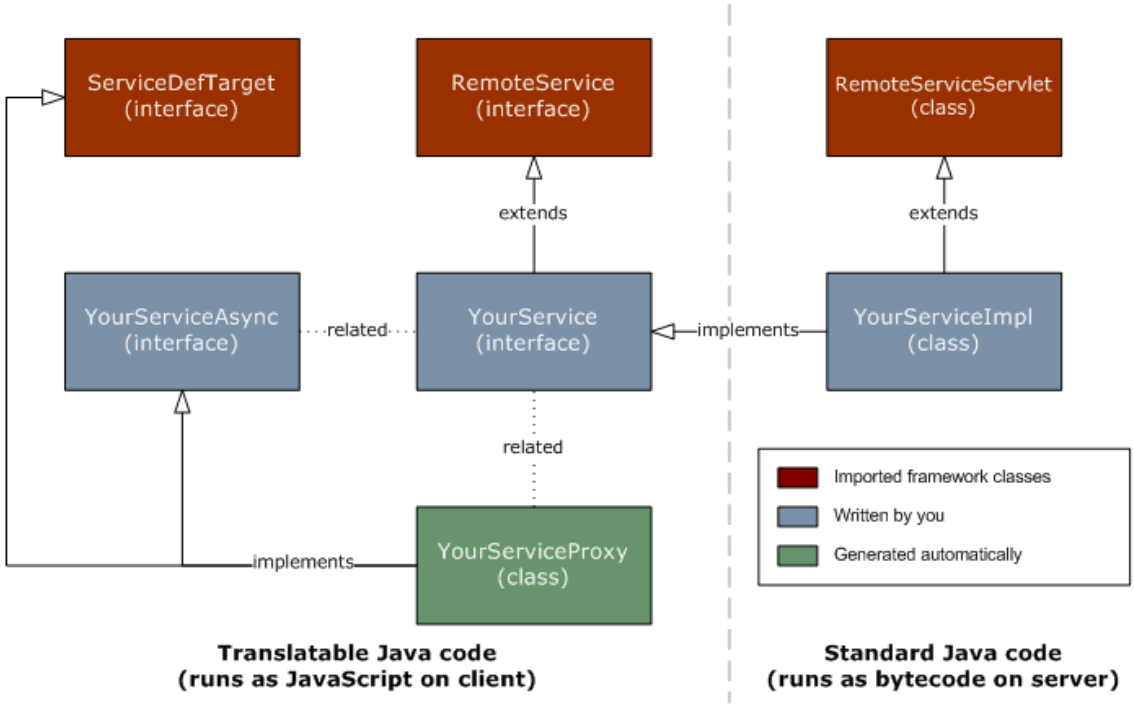

Figura 16: Struttura GWT RPC

Per ogni servizio offerto dal server devono essere dichiarate:

- Un'interfaccia che estende `RemoteService`, all'interno della quale ci sarà la dichiarazione del servizio offerto dal server (la dichiarazione del metodo RPC);
- Una classe per l'implementazione del codice lato server che estende `RemoteServiceServlet` e implementa l'interfaccia creata in precedenza.

[17] La serializzazione è un'attività che permette di salvare un oggetto in uno stream di byte che successivamente può essere salvato in maniera persistente in un file oppure inviato sulla rete

- Un'interfaccia asincrona, basata sulla definizione dell'interfaccia sincrona corrispondente, utilizzata per richiamare il servizio lato client.

La relazione fra l'interfaccia sincrona, in cui è definito il servizio, e la sua controparte asincrona deve seguire determinate regole. Infatti i nomi delle interfacce devono seguire una politica standard definita da GWT.

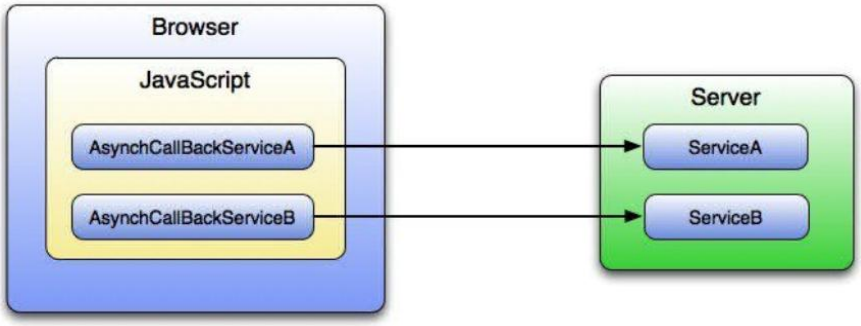

Figura 17: Invocazione servizio

Di seguito viene mostrata la definizione di una tipica interfaccia sincrona *client-side*. Come si nota dal codice, l'interfaccia estende `RemoteService` e contiene al suo interno la definizione di un metodo RPC.

```
package com.example.foo.client;

import com.google.gwt.user.client.rpc.RemoteService;

public interface MyService extends RemoteService {
  public String myMethod(String s);
}
```

Ogni implementazione del servizio definito, dovrà estendere RemoteServiceServlet e dare corpo al metodo RPC definito nell'interfaccia sincrona appena descritta. Una tipica implementazione è la seguente:

```
package com.example.foo.server;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import com.example.client.MyService;


public class MyServiceImpl extends RemoteServiceServlet implements
    MyService {

  public String myMethod(String s) {
    // Do something interesting with 's' here on the server.
    return s;
  }
```

```
}
```

Per poter richiamare il metodo RPC, è necessario creare un'interfaccia asincrona basata sulla definizione di quella sincrona originaria. Seguendo gli esempi precedenti, la definizione sarà la seguente:

```
package com.example.foo.client;

interface MyServiceAsync {
  public void myMethod(String s, AsyncCallback<String> callback);
}
```

L'interfaccia asincrona deve essere identica a quella sincrona, salvo che, per ciascuno dei metodi che dichiara, deve contenere il parametro aggiuntivo `AsyncCallback` e questi generalmente non dovrebbero presentare valore di ritorno. Ma è facile intuirne il perché: per definizione, in un sistema asincrono, i client non devono rimanere "bloccati" in attesa di un valore di ritorno. Quindi, dopo che è stata effettuata una chiamata asincrona, tutte le eventuali risposte del server devono essere rese attraverso l'oggetto `AsyncCallback`.

Si è detto che la comunicazione tra il server e il client avviene attraverso lo scambio di oggetti. Sia il server che il client infatti, sono in grado di serializzare e deserializzare gli oggetti che inviano/ricevono. GWT pone dei limiti alle tipologie di oggetti che è possibile far scambiare ad un server ed un client.

Gli oggetti ammessi sono i seguenti:

- Tipi primitivi Java: `boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, `short`;
- I *wrapper*[18] dei tipi primitivi;
- Un sottoinsieme degli oggetti appartenenti a JRE;
- Qualsiasi tipo definito dall'utente purché che implementi l'interfaccia `Serializable`;

[18] Tutti i tipi primitivi hanno delle controparti nelle classi, per esempio la classe `Integer` corrisponde al tipo primitivo `int`. Questo genere di classi prende il nome di *wrapper*.

Capitolo 3

VOGCLUSTERS

3.1 DATA LAYER

3.1.1 IL DATABASE

Il database si struttura in 18 entità e 25 relazioni di cui 8 vengono convertite in altrettante tabelle, seguendo il principio per cui le relazioni M:N necessitano obbligatoriamente della costituzione di un'ulteriore entità. Quest'ultima comprenderà le chiavi primarie delle entità messe in relazione ed eventuali attributi aggiuntivi della relazione stessa.

Per questo progetto è stata utilizzata la versione 5.0.45 di MySQL su una piattaforma GNU/Linux.

3.1.1.1 DESCRIZIONE DELLE ENTITÀ

Di seguito verrà riportata una descrizione dettagliata di ogni singola tabella con riferimento al ruolo che essa gioca all'interno dello schema del database.

- **object**: questa tabella viene utilizzata per identificare gli oggetti gestiti dalla web application. Con il termine "oggetto" si intende un ammasso globulare, una pulsar o una stella del ramo orizzontale.
- **gcluster**: questa tabella viene utilizzata per descrivere un ammasso globulare galattico o extragalattico.
- **pulsar**: questa tabella viene utilizzata per descrivere un oggetto pulsar presente all'interno di un ammasso globulare.
- **star**: questa tabella, analoga alla precedente, viene utilizzata per descrivere una stella del ramo orizzontale presente all'interno di un ammasso globulare.
- **gcAttribute**: questa tabella contiene tutte le informazioni relative agli attributi degli ammassi globulari.

- **plsAttribute**: questa tabella contiene tutte le informazioni relative agli attributi delle pulsar.
- **starAttribute**: questa tabella, analoga alle precedenti, contiene tutte le informazioni relative agli attributi delle stelle del ramo orizzontale.
- **source**: questa tabella viene utilizzata per identificare le risorse da cui provengono le informazioni relative agli oggetti.
- **user**: questa tabella contiene tutte le informazioni riguardanti gli utenti abilitati.
- **catalogue**: questa tabella contiene tutte le informazioni riguardanti i cataloghi da cui vengono prelevate parte delle informazioni visualizzate dalla web application.
- **session**: questa tabella viene utilizzata per tenere memoria delle sessioni utente.
- **image**: questa tabella viene utilizzata per contenere tutte le informazioni riguardanti le immagini e i diagrammi associati ai vari oggetti.
- **tag:** questa tabella contiene la lista dei tag utilizzati.
- **biblioNotes**: questa tabella contiene le note e i riferimenti associati agli oggetti oppure ai singoli attributi relativi agli oggetti stessi.
- **notes**: questa tabella contiene i commenti inseriti dai vari utenti.
- **biblioRef**: questa tabella contiene i riferimenti bibliografici (compresi riferimenti a particolari articoli presenti in una rivista).
- **author**: questa tabella contiene le informazioni riguardanti gli autori delle riviste e dei cataloghi.
- **paper**: questa tabella contiene le informazioni riguardanti le riviste.

3.1.1.2 SCHEMA ENTITÀ-RELAZIONE

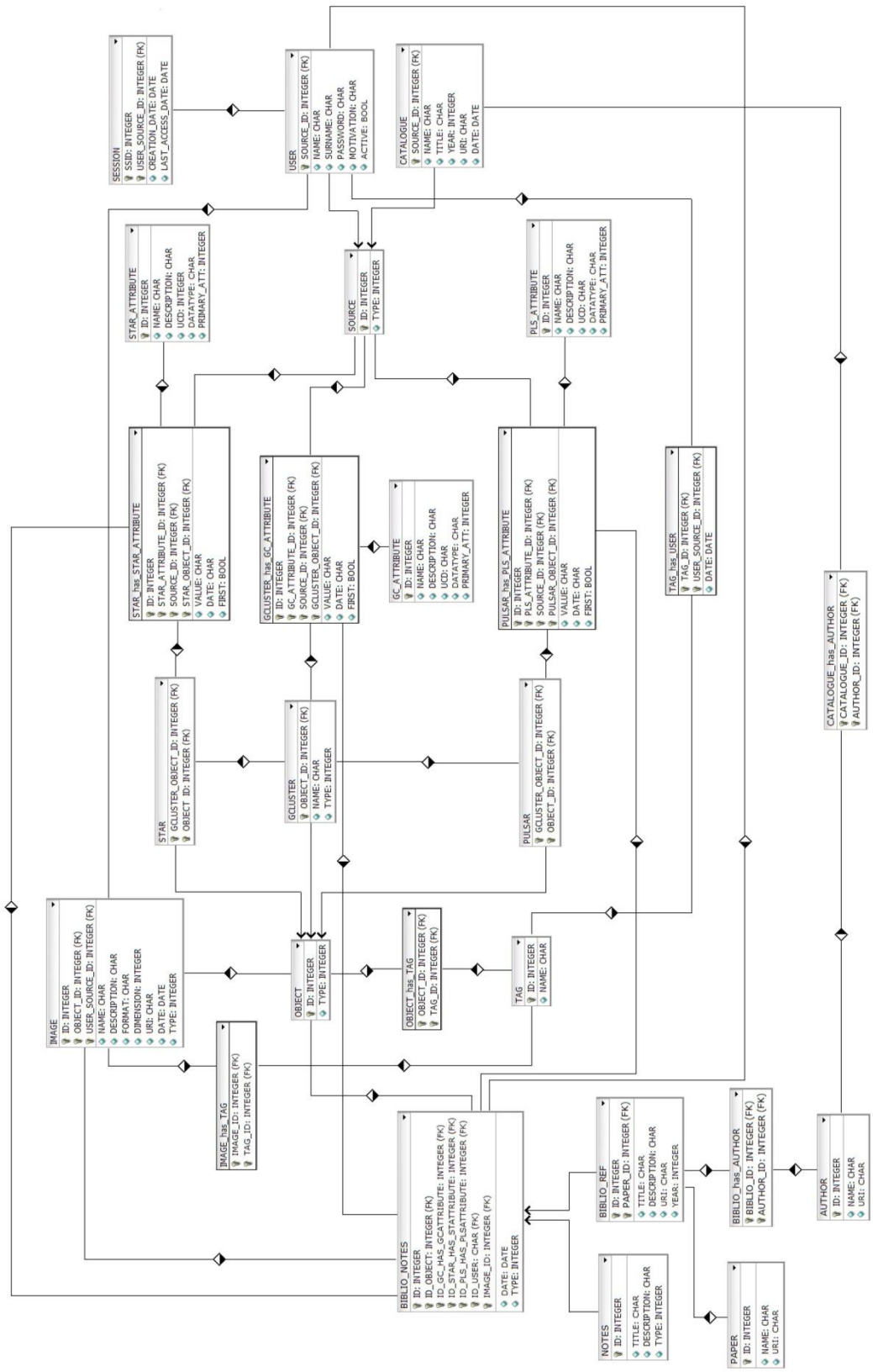

Figura 18: Schema Entità-Relazione

3.2 DATA ACCESS AND PROCESS LAYER

3.2.1 COMPONENTE VOGCACCESS

Questa componente appartiene al *Data Access and Process Layer* e si occupa dell'accesso e della modifica dei dati permanenti.

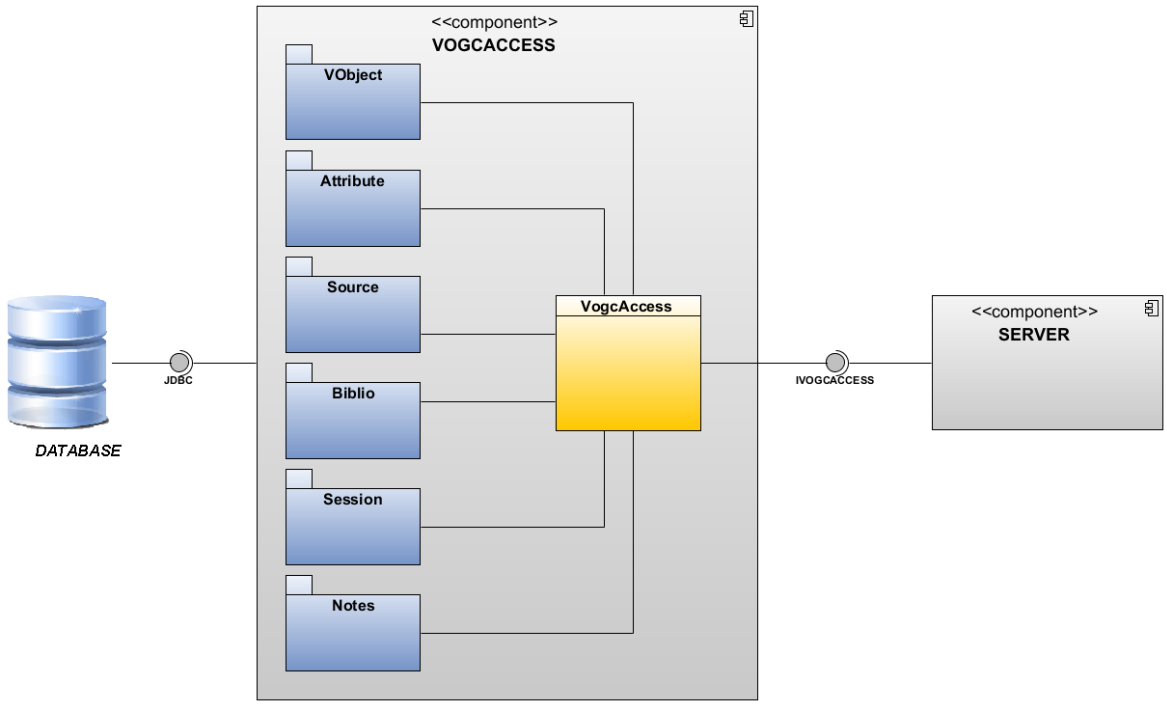

Figura 19: Componente VOGCACCESS

Come si nota dalla Figura 19, la componente in questione accede ai metodi forniti dall'interfaccia JDBC per la comunicazione con il database. L'interfaccia *IVOGCACCESS* invece contiene le dichiarazioni di tutti i metodi per poter accedere alle informazioni presenti nelle singole tabelle del database, cioè metodi per effettuare inserimento, cancellazione e modifica delle tuple. La classe *VogcAccess* implementa l'interfaccia e realizza tali i metodi.

In totale sono state sviluppate 29 classi. 27 di queste sono della tipologia *Class**TableName**eTable*. Con questa nomenclatura si vuole indicare una classe che contiene metodi opportuni per effettuare l'inserimento, la cancellazione o la modifica di una tupla nella tabella ***TableName*** del database.

Due sono le classi fondamentali all'interno di questa componente: *ConnectionManager* e *DBStatement*. La prima viene utilizzata per la connessione al database e la seconda per la creazione delle query SQL (Figura 20).

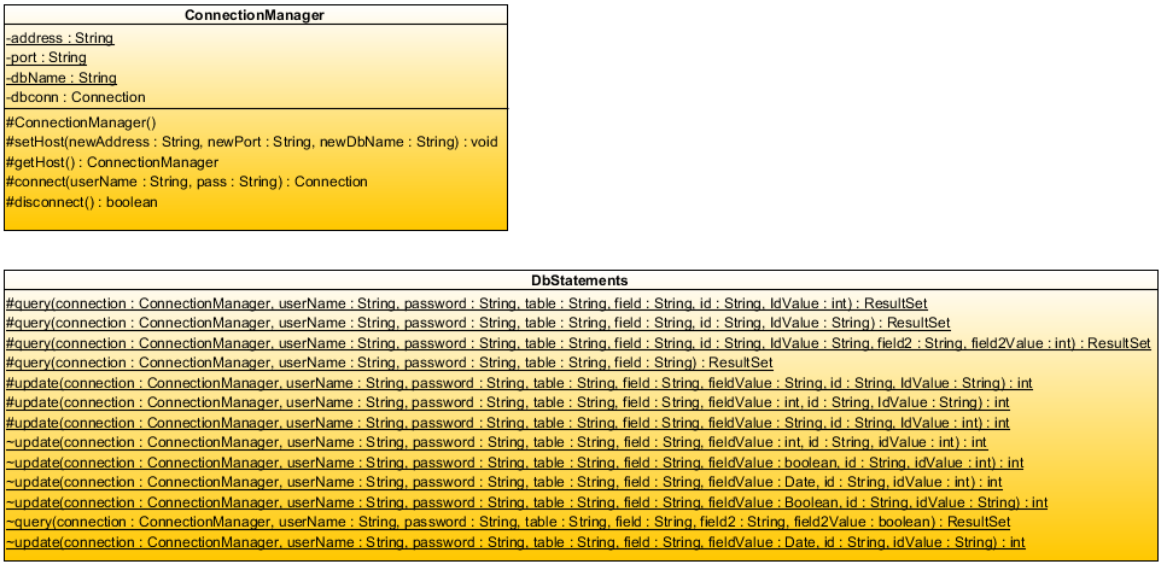

Figura 20: Classi Connection Manager e DBStatement

Di seguito verranno analizzati nel dettaglio i package *Vobject*, *Source*, *Biblio*, *Session* e *Notes*.

3.2.1.1 VOBJECT PACKAGE

In questo package sono presenti le classi che manipolano i dati relativi a ammassi globulari e oggetti ad essi annessi. Tali dati sono quelli presenti nelle tabelle *vobject*, *gcluster*, *star*, *pulsar*, *gcluster_has_attribute*, *star_has_attribute*, *pulsar_has_attribute* e *vobject_has_tag*.

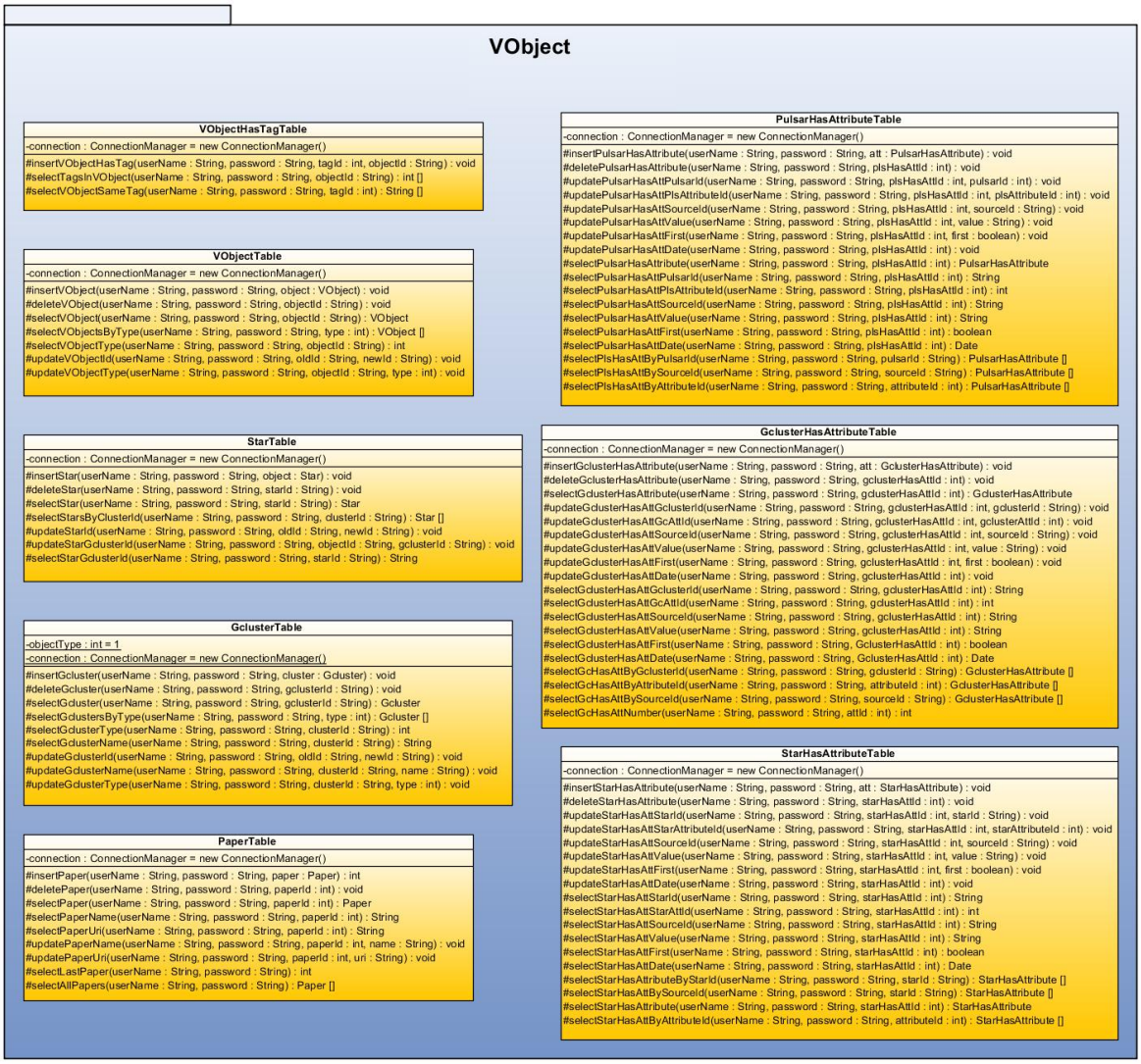

Figura 21: VObject Package

3.2.1.2 ATTRIBUTE PACKAGE

In questo package sono presenti le classi che manipolano i dati relativi agli attributi da associare ai vari oggetti, cioè i dati presenti nelle tabelle *gc_attribute*, *pls_Attribute* e *star_attribute.*

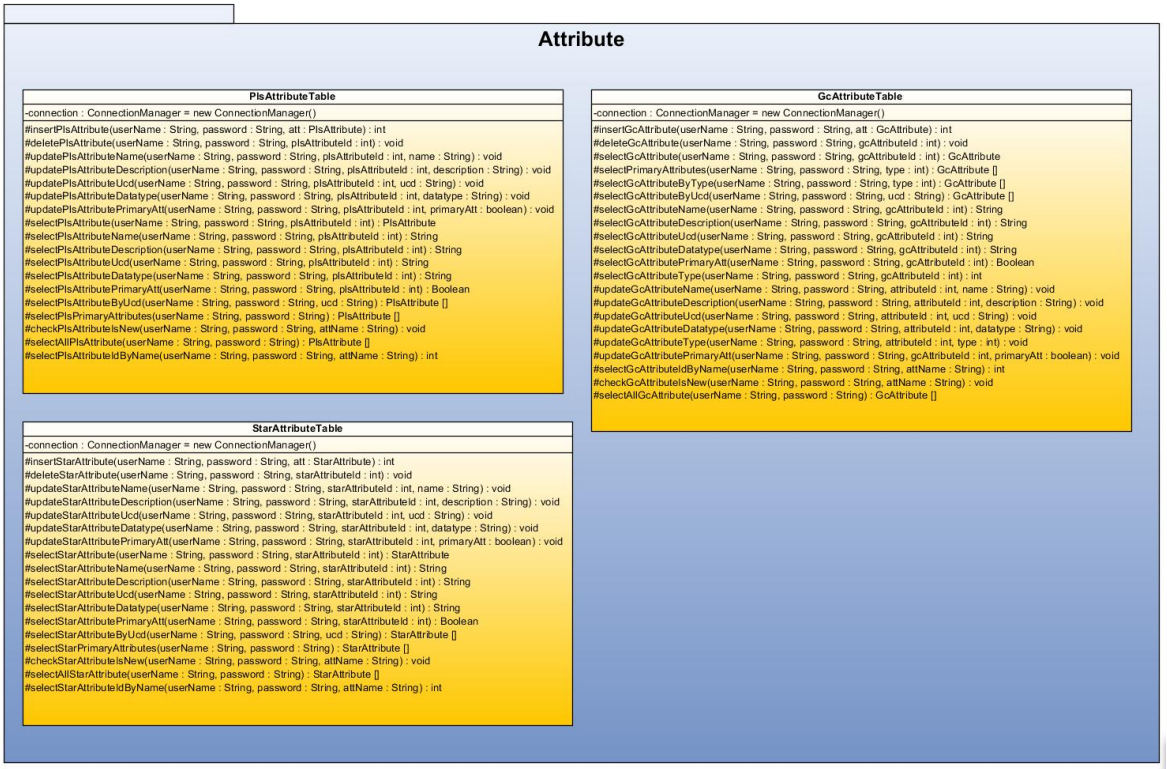

Figura 22: Attribute Package

3.2.1.3 BIBLIO PACKAGE

In questo package sono presenti le classi che manipolano i dati relativi alle note e ai riferimenti bibliografici, cioè quelli presenti nelle tabelle *biblio_ref*, *biblio_notes*, *paper*, *author* e *biblio_has_author*.

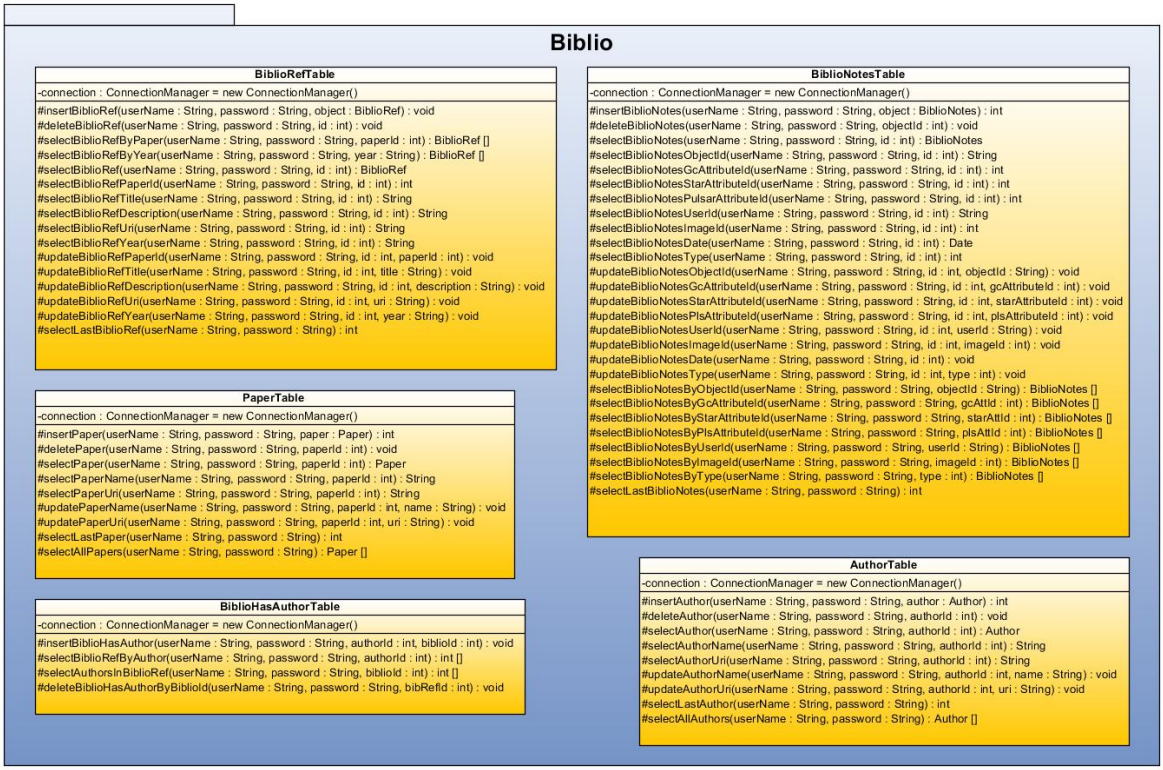

Figura 23: Biblio Package

3.2.1.4 SOURCE PACKAGE

In questo package sono presenti le classi che manipolano i dati relativi alle risorse utilizzate, cioè quelli presenti nelle tabelle *source*, *user*, *catalogue* e *catalogue_has_author*. Ogni informazione relativa ad un oggetto può provenire sia da un catalogo astronomico oppure può essere inserita dagli utenti DAME abilitati. Le entità utente e catalogo appartengono entrambe alla categoria "risorsa".

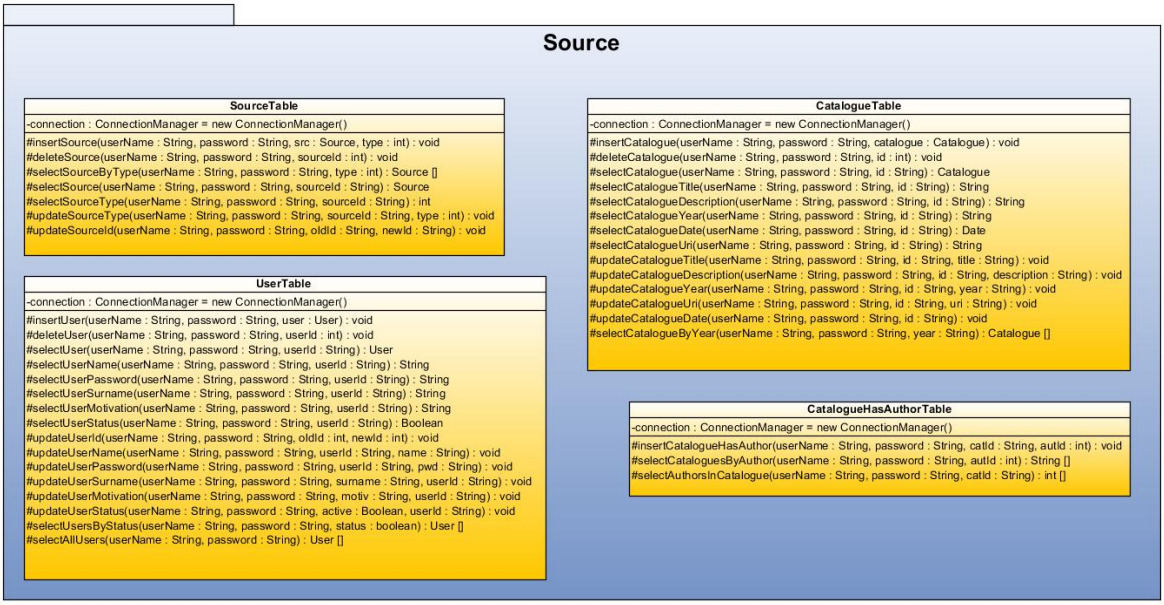

Figura 24: Source Package

3.2.1.5 SESSION PACKAGE

In questo package è presente la classe *SessionTable* utilizzata per manipolare le informazioni relative alle sessioni degli utenti, cioè quelle presenti nella tabella *session* del DB.

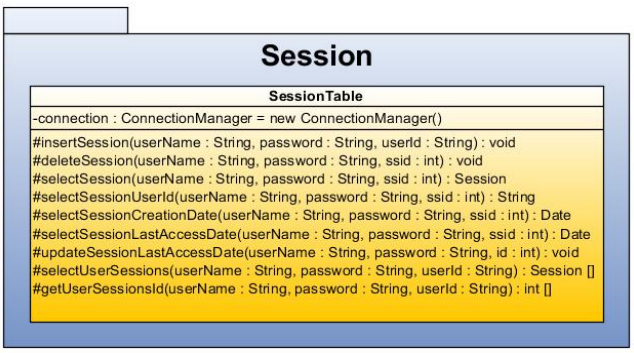

Figura 25: Session Package

3.2.1.6 NOTES PACKAGE

In questo package sono presenti le classi che manipolano i dati relativi alle immagini, alle note e ai tag. Note e tag possono essere associate sia alle immagini che agli oggetti e vengono inseriti dagli utenti DAME abilitati. Queste classi agiscono sulle tabelle *notes*, *image*, *tag*, *tag_has_user*, *image_has_tag*.

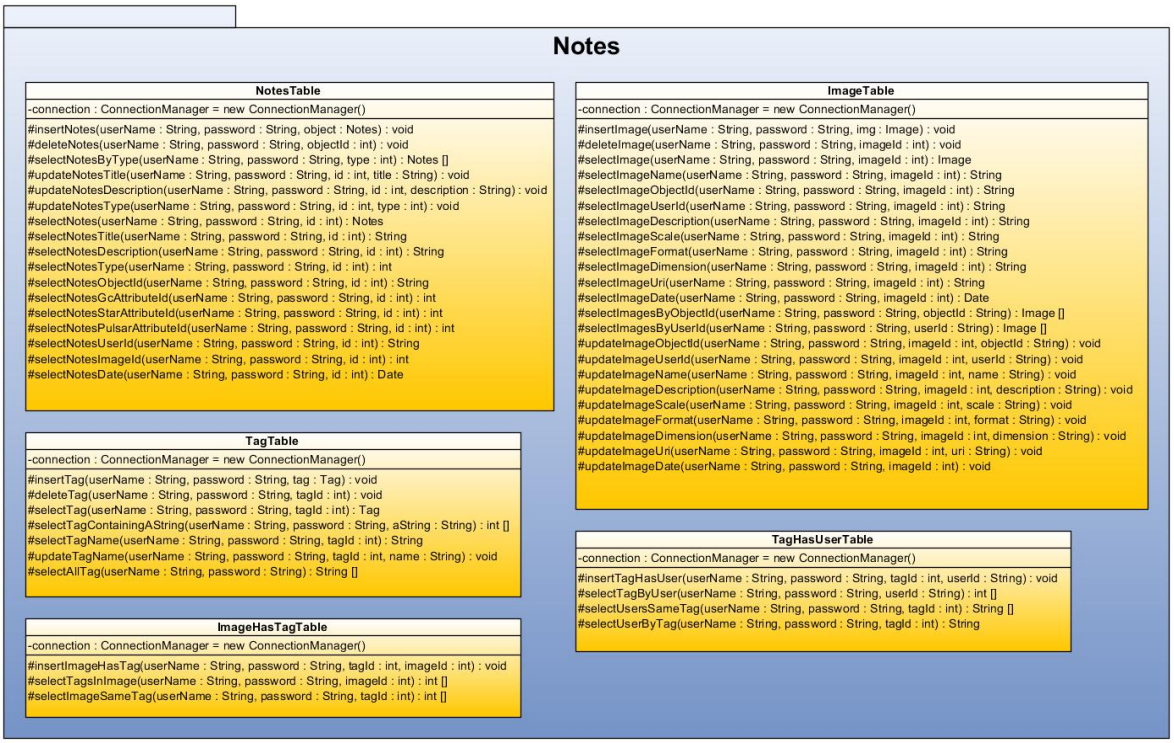

Figura 26: Notes Package

3.3 SERVICE LAYER

3.3.1 COMPONENTE SERVER

Il server si occupa di ricevere le richieste provenienti dal *FrontEnd Layer*, di elaborarle e di comunicare con il *Data Access and Process Layer* per salvare o recuperare i dati presenti nel database.

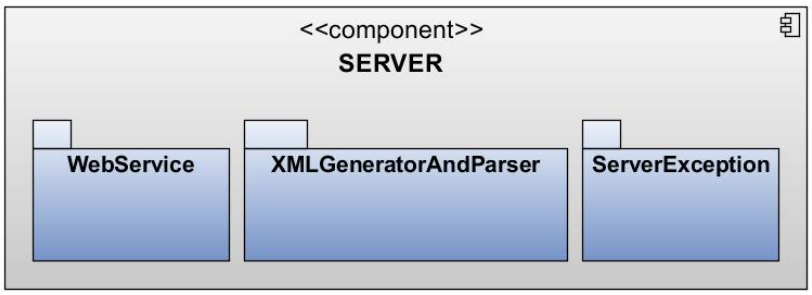

Figura 27: Componente SERVER

Come si nota dalla Figura 27, nella componente SERVER sono presenti tre package: *WebService*, *XMLGeneratorAndParser* e *ServerException*. Nel primo sono presenti le servlet che si occupano di ricevere ed elaborare le richieste provenienti dal *FrontEnd Layer*. Nel secondo package sono presenti le classi responsabili della generazione e del *parsing* dei documenti XML. Infine nell'ultimo package è presente la classe *Messages* che si occupa della gestione delle stringhe di errore.

3.3.1.1 WEBSERVICE PACKAGE

In questo package (Figura 28) sono presenti le servlet che gestiscono la comunicazione *Service-FrontEnd Layers*. Per una miglior comprensione, queste sono state suddivise in cinque ulteriori package. Procederemo con l'analisi singola di ognuno di essi. Nell'analisi delle funzionalità offerte dalle singole servlet si è scelto di offrire una descrizione più dettagliata solo laddove ritenuto necessario.

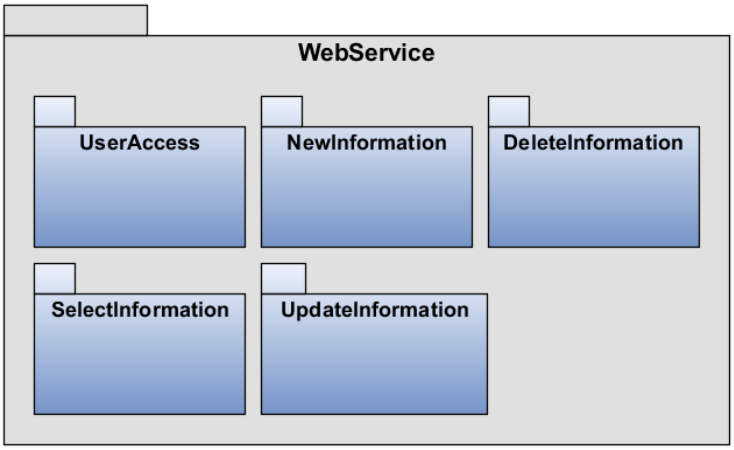

Figura 28: WebService Package

3.3.1.1.1 USERACCESS PACKAGE

In questo package sono presenti tutte le servlet che gestiscono le richieste di autenticazione e abilitazione degli utenti.

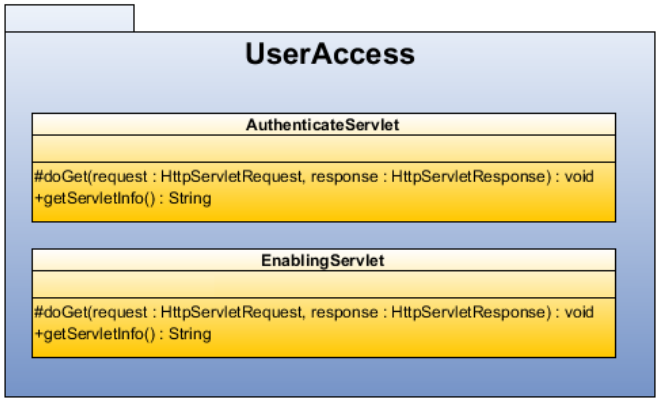

Figura 29: UserAccess Package

- ***AuthenticateServlet***: servlet invocata per stabilire se l'utente loggato in DAME è un utente abilitato in VOGCLUSTERS o meno. Può essere invocata da qualsiasi tipologia di utente in quanto viene utilizzata proprio per stabilirne i privilegi all'interno della web application.

- ***EnablingServlet***: questa servlet viene invocata in seguito ad una richiesta di abilitazione da parte di un *utente DAME*. Riceve come parametri di *request* le informazioni descrittive dell'utente. Elabora i dati, se sono corretti, viene inviata un'e-mail all'amministratore del sistema con il riepilogo dei dati. Sarà poi l'amministratore, in un momento successivo, a registrare l'utente nel DB locale come *utente DAME abilitato*.

3.3.1.1.2 NEW INFORMATION PACKAGE

In questo package sono presenti tutte le servlet che gestiscono le richieste di inserimento di nuove informazioni nel database.

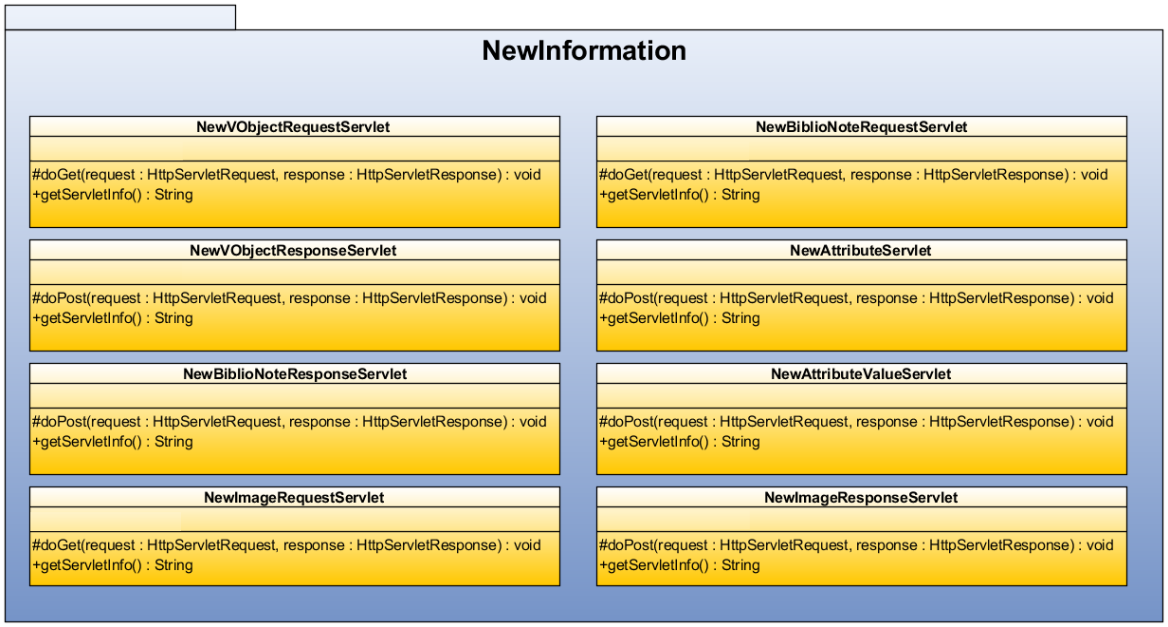

Figura 30: NewInformation Package

- **Richiesta Inserimento Nuovo Oggetto (*NewVObjectRequestServlet*):** servlet invocata in seguito ad una richiesta di inserimento nuovo oggetto (sia esso un ammasso globulare, una pulsar o una stella del ramo orizzontale). Viene utilizzata per restituire al client la lista degli attributi primari relativi all'oggetto. Lo scopo di questa servlet è garantire all'utente di VOGCLUSTERS una vista sempre aggiornata degli attributi relativi agli oggetti.

- **Esito Inserimento Nuovo Oggetto (*NewVObjectResponseServlet*):** questa servlet viene invocata in seguito ad una richiesta di salvataggio dovuta all'inserimento di un nuovo oggetto.

- **Inserimento nuovo attributo (*NewAttributeServlet*):** servlet invocata in seguito ad una richiesta di inserimento nuova tipologia di attributo (sia esso relativo ad un ammasso globulare, ad una pulsar o ad una stella).

- **Inserimento nuovo valore attributo (*NewAttributeValueServlet*):** questa servlet viene invocata quando l'*utente DAME abilitato* desidera inserire un nuovo valore per un attributo esistente.

- **Richiesta Inserimento Nuova Nota Bibliografica (*NewBiblioNoteRequestServlet*):** questa servlet viene invocata in seguito alla richiesta di inserimento di una nuova nota, sia che essa sia un semplice commento di un utente, sia che essa sia un nuovo riferimento bibliografico. Non riceve nessun parametro di input ma si limita a generare un documento XML contenente tutte le informazioni utili (Riviste, Autori) all'utente per poter effettuare l'inserimento.

- **Esito Inserimento Nuova Nota Bibliografica (*NewBiblioNoteResponseServlet*):** servlet invocata in seguito ad una richiesta di salvataggio dovuta all'inserimento di un nuova nota bibliografica.

- **Richiesta Inserimento Nuova/o Immagine/Grafico (*NewImageRequestServlet*):** questa servlet viene invocata in seguito ad una richiesta di inserimento di una nuova immagine o grafico da parte dell'utente. Non riceve nessun parametro di *request*, ma si limita a generare la lista dei tag disponibili nel DB. Qualora il tag desiderato non sia presente nella lista, una procedura a parte consentirà all'utente di aggiungerlo.

- **Esito Inserimento Nuova/o Immagine/Grafico (*NewImageResponseServlet*):** questa servlet viene invocata in seguito ad una richiesta di salvataggio di una nuova immagine (o grafico).

- **Inserimento Nuovo Tag (*NewTagServlet*):** servlet invocata in seguito ad una richiesta di inserimento di un nuovo tag.

3.3.1.1.3 DELETEINFORMATION PACKAGE

In questo package sono presenti tutte le servlet che gestiscono le richieste di eliminazione di informazioni o risorse.

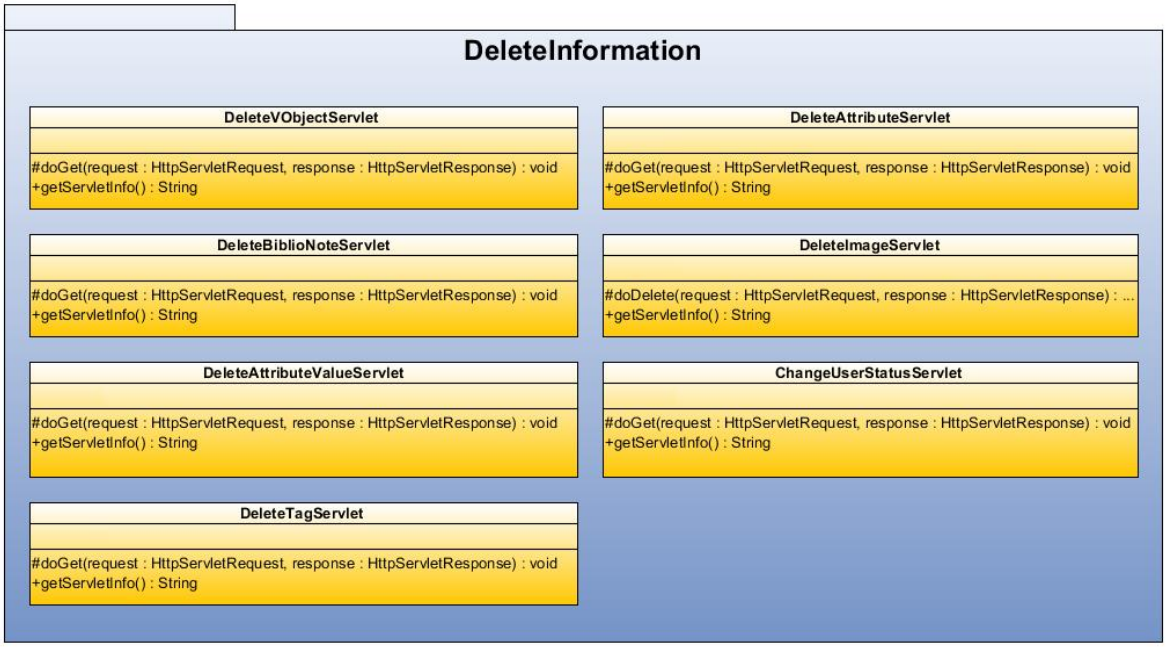

Figura 31: DeleteInformation Package

- **Eliminazione Attributo (*DeleteAttributeServlet*):** Questa servlet viene invocata in seguito ad una richiesta di eliminazione attributo. Riceve come parametro di input lo *userId* dell'utente connesso, l'*ID* dell'oggetto di cui si desidera eliminare un attributo e l'*ID* dell'attributo da eliminare. Dopo aver effettuato il controllo sulla validità di tali parametri si procede a verificare che l'utente possegga i privilegi per poter effettuare tale operazione. In caso affermativo, l'attributo viene eliminato altrimenti viene generato un messaggio di errore.

- **Eliminazione Valore Attributo (*DeleteAttributeValueServlet*):** Questa servlet viene invocata in seguito ad una richiesta di eliminazione del valore di un particolare attributo. Riceve come parametri di *request* l'*ID* dell'utente connesso, l'*ID* dell'oggetto e l'*ID* del particolare valore da eliminare (l'*ID* del valore è differente dall'*ID* dell'attributo: in questo caso si vuole eliminare un particolare valore, non l'intero attributo). Dopo aver effettuato il controllo sulla validità di tali parametri si procede a verificare che l'utente possegga i privilegi per poter effettuare tale operazione. In caso affermativo, il valore viene eliminato altrimenti viene generato un messaggio di errore.

- **Eliminazione Oggetto (*DeleteVobjectServlet*):** Questa servlet viene invocata in seguito ad una richiesta di eliminazione oggetto (ammasso, pulsar o stella). Riceve come parametri di *request* l'*ID* dell'oggetto. Dopo aver effettuato il controllo sulla validità del parametro si procede con il verificare che l'oggetto in questione non presenti nessun attributo, nessuna nota e più in generale, che l'oggetto non presenti nessuna informazione aggiuntiva ad esso associato. Infatti, questa servlet viene invocata solo nel caso in cui l'utente abbia eliminato tutte le informazioni associate ad un oggetto da lui stesso inserito. Se non si presentano condizioni di errore, l'oggetto viene eliminato dal DB, altrimenti viene generato un messaggio di errore.

- **Eliminazione Immagine (*DeleteImageServlet*):** Questa servlet viene invocata in seguito ad una richiesta di eliminazione immagine. Prende in input l'*ID* dell'immagine e lo *userId* dell'utente connesso. Dopo aver verificato che i parametri siano corretti e che l'utente possegga i privilegi per poter effettuare l'operazione, si procede con l'eliminazione della risorsa dal server.

- **Attivazione/Disattivazione Utente (*ChangeUserStatus*):** Questa servlet viene invocata in seguito ad una richiesta di attivazione o disattivazione di un utente. Prende come parametri di *request* l'*ID* dell'utente e una *flag* che indica il tipo di operazione che si desidera effettuare (attivazione o disattivazione). Se i parametri sono corretti verrà prodotto un messaggio di successo, altrimenti un messaggio di errore.

- **Eliminazione Tag (*DeleteTagServlet*)**: Questa servlet viene invocata in seguito ad una richiesta di eliminazione tag. Riceve come parametro di *request* l'*ID* del tag da eliminare e lo *userId* dell'utente che desidera effettuare l'operazione. Se l'utente possiede i privilegi per poter effettuare l'operazione, il tag viene eliminato e viene generato un messaggio di successo, altrimenti viene generato un messaggio di errore.

3.3.1.1.4 SELECTINFORMATION PACKAGE

In questo package sono presenti tutte le servlet che vengono invocate in seguito a richieste di visualizzazione di informazioni.

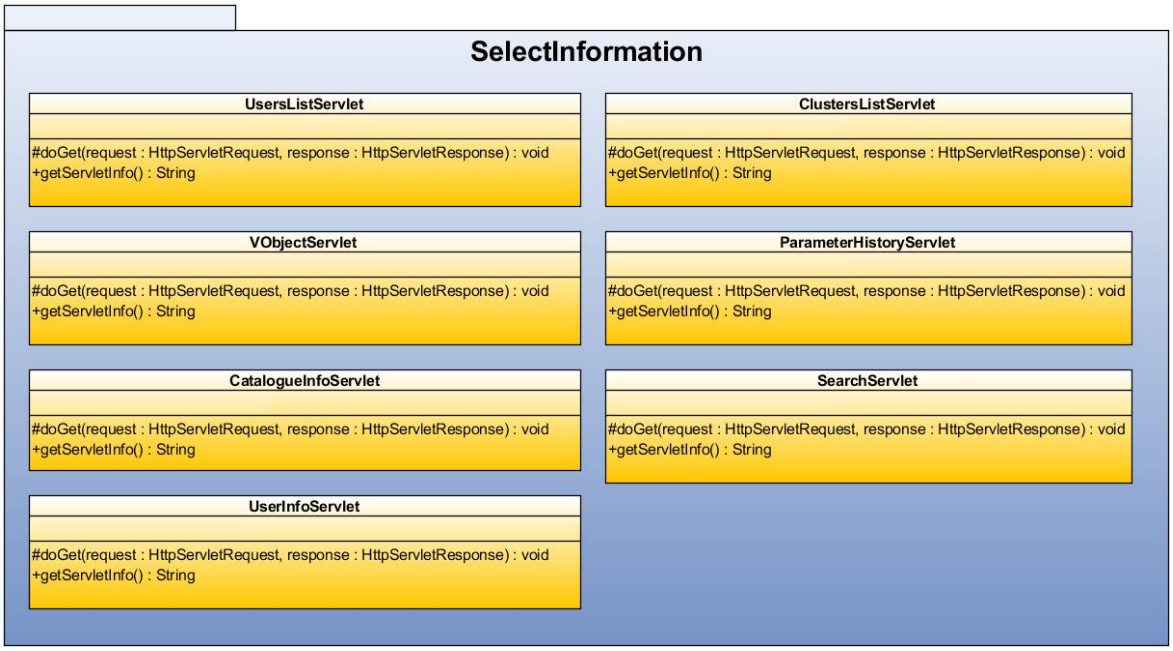

Figura 32: SelectInformation Package

- **Lista Ammassi Globulari (*ClustersListServlet*):** Questa servlet viene invocata quando si richiede la lista degli ammassi globulari presenti in nel DB di VOGCLUSTERS.

- **Visualizzazione informazioni oggetto (*VobjectServlet*):** Questa servlet può essere invocata quando viene richiesta la visualizzazione delle informazioni relative ad un ammasso globulare, ad una pulsar oppure per le informazioni relative ad una stella.

- **Visualizza Storia Parametro (*ParameterHistoryServle*):** Questa servlet viene invocata in seguito ad una richiesta di visualizzazione della storia delle osservazioni relative ad un singolo parametro (sia che esso appartenga ad un ammasso globulare, ad una pulsar oppure ad una stella).

- **Visualizza Lista Utenti (*UsersListServlet*):** Questa servlet viene invocata in seguito ad una richiesta di visualizzazione della lista degli utenti DAME abilitati.

- **Visualizza Informazioni Catalogo (*CatalogueInfoServlet*):** Questa servlet viene invocata in seguito ad una richiesta di visualizzazione delle informazioni relative ad un catalogo.
- **Visualizza Informazioni Utente (*UserInfoServlet*):** Questa servlet viene invocata in seguito ad una richiesta di visualizzazione delle informazioni relative ad un singolo utente. Può essere invocata dall'utente DAME abilitato e dall'utente DAME amministratore.
- **Ricerca *(SearchServlet)***: Questa servlet viene invocata quando l'utente effettua una ricerca per tag. Infatti ad ogni oggetto o immagine l'utente ha la possibilità di associare tag utilizzati per facilitarne la ricerca.

3.3.1.1.5 UPDATEINFORMATION PACKAGE

In questo package sono presenti le servlet che gestiscono le richieste di aggiornamento delle informazioni presenti nel DB locale. Tutte le servlet in questo package, come quelle descritte in precedenza, ricevono sempre come parametro di request l'*ID* dell'utente che richiede l'operazione e i dati utili all'aggiornamento.

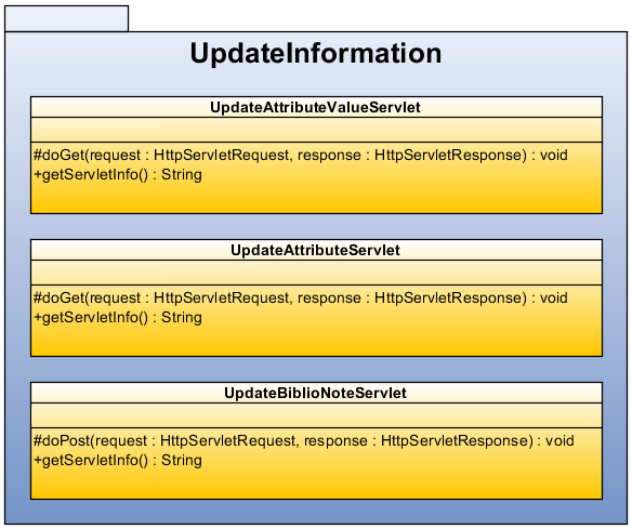

Figura 33: UpdateInformation Package

- **Aggiornamento informazioni attributo (*UpdateAttributeServlet*):** servlet invocata in seguito ad una richiesta di aggiornamento delle informazioni relative a un attributo.

- **Aggiornamento valore attributo (*UpdateAttributeValueServlet*):** servlet invocata in seguito ad una richiesta di aggiornamento del valore di un attributo.
- **Aggiornamento Nota Bibliografica (*UpdateBiblioNotesServlet.java*):** servlet invocata per aggiornare i dati relativi ad una nota bibliografica.

3.3.1.2 XMLPARSERANDGENERATOR PACKAGE

In questo package sono presenti due classi: *XMLGenerator* e *XMLParser*. La prima è responsabile della creazione dei documenti XML che ogni singola servlet invierà al *FrontEnd Layer*. La seconda invece ha il compito opposto: si occupa del *parsing* degli XML ricevuti dalle servlet.

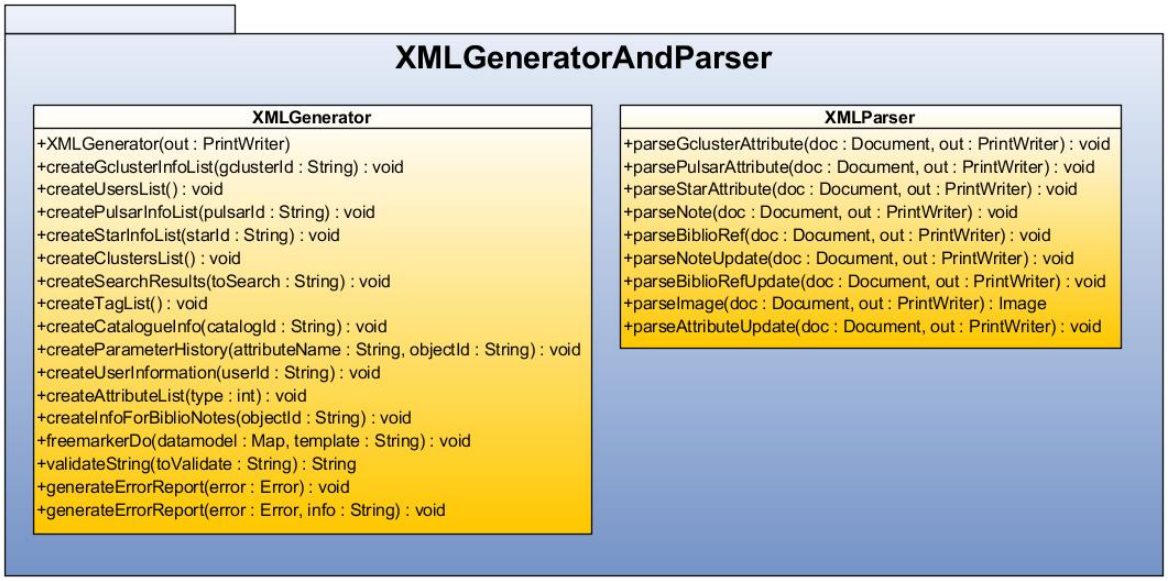

Figura 34: XMLGeneratorAndParser Package (Service Layer)

Nella classe *XMLGenerator* vengono utilizzate le librerie di FreeMarker, mentre in *XMLParser* le librerie di JDOM. Entrambe le tecnologie sono state ampiamente discusse nei capitoli precedenti.

3.4 FRONTEND LAYER

3.4.1 COMPONENTE WEB APP VOGC

Questa componente, sviluppata con il framework GWT, ha il compito di interagire sia con l'utente finale che con le componenti del *Service Layer*, in particolare con le servlet presenti nel package *Web Service*.

Ogni qual volta la WEB APP vuole recuperare delle informazioni, effettua delle richieste HTTP al Server. La componente oggetto di questo paragrafo possiede la logica che consente di effettuare le richieste HTTP al Server. Ogni richiesta, identificata attraverso un URL, rappresenta un particolare caso d'uso, cioè un particolare servizio esposto dal Server.

Come si nota dalla Figura 35, all'interno della componente WEB APP VOGC sono presenti cinque package: *HTTPRequest*, *HTTPRequester*, *Service Declaration*, *ServiceImplementation* e *XMLGeneratorAndParser*. Nei successivi paragrafi procederemo con l'analisi singola di ognuno di essi.

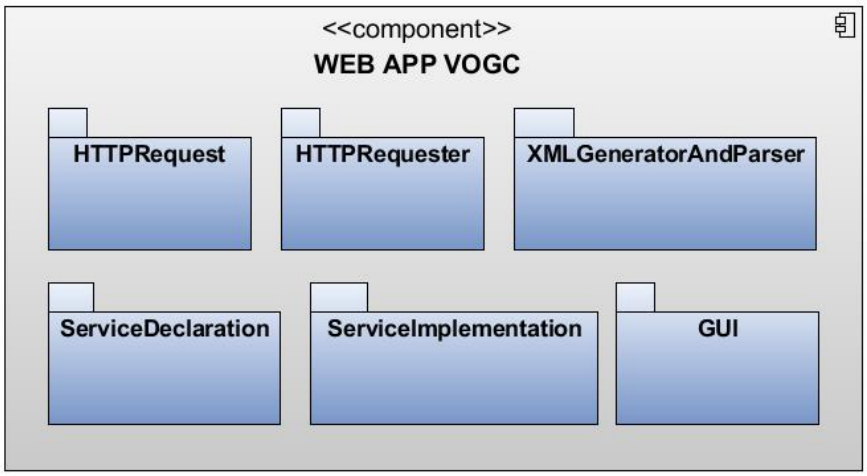

Figura 35: Componente WEB APP VOGC

Lo scopo delle librerie *HTTPRequest* e *HTTPRequester* è quello di dare la possibilità a chiunque in futuro di realizzare la propria implementazione della WEB APP. Lo sviluppatore infatti, attraverso i metodi forniti da *HTTPRequest*, potrà scegliere quali servizi del Server invocare. L'invocazione di un particolare servizio avverrà attraverso la libreria *HTTPRequester*. Quindi lo sviluppatore non dovrà preoccuparsi dell'interazione con il Server, ma dovrà solamente gestire i documenti XML di risposta. In altre parole, dovrà preoccuparsi di dare una diversa implementazione delle librerie *Service Declaration* e *ServiceImplementation*, che, nel nostro caso, sono state scritte seguendo la politica di GWT, ma nulla vieterà di implementarle diversamente.

3.4.1.1 HTTPREQUEST PACKAGE

In questo package è presente un'unica classe: *VogcServerUseCase*. Questa contiene tanti metodi, quanti sono i servizi offerti dal *Service Layer*. Ognuno dei metodi presenti ritorna l'URL del servizio richiesto.

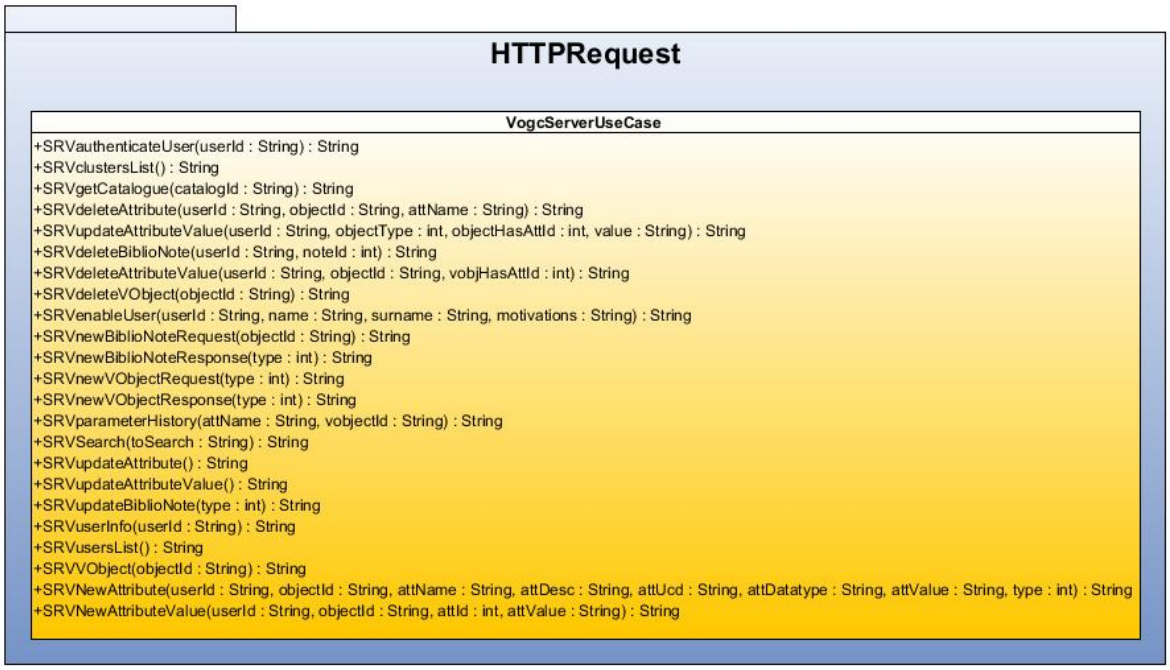

Figura 36: HTTPRequest Package

Di seguito viene riportato l'elenco degli URL utilizzati per poter effettuare richieste al server.

USE CASE	SERVER URI
Autenticazione utente	`GET/autenticate?userId=x`
Abilitazione utente	`GET/enable?userId=x`
Visualizza ammasso	`GET/vobject?objectId=x`
Visualizza catalogo	`GET/catalogue?catId=x`
Salva nota bibliografica	`POST/saveNote?type=x`
Salva oggetto	`POST/saveVobject?type=x`

Ricerca oggetto	GET/search?string=x
Inserimento nuovo attributo	GET/newAttribute?userId=x&objectId=y&attName=z&attDesc=t&attUcd=q&attDatatype=r&attValue=s&type=k
Inserimento nuovo valore attributo	GET/newValue?userId=x&objectId=y&attId=z&attValue=t
Aggiorna attributo	POST/updateAtt
Aggiorna nota bibliografica	POST/updateNote?type=x
Aggiorna valore attributo	GET/updateValue?userId=x&objectType=y&objectHasAttId=t&value=z
Richiedi lista attributi	GET/attList?type=x
Visualizza lista ammassi	GET/clusters
Visualizza lista utenti	GET/users
Cancella attributo	GET/deleteAttribute?userId=x&objectId=y&attName=z
Cancella nota bibliografica	GET/ deleteBiblioNote?userId=x&noteId=y;
Cancella valore attributo	GET/deleteValue?userId=x&objectId=y&vobjHasAttId=z;
Cancella ammasso	GET/deleteVObject?objectId=x
Cancella immagine	DELETE/deleteImage?imageId=x
Recupera informazioni per note bibliografiche	GET/ newNoteRequest?objectId=x

3.4.1.2 HTTPREQUESTER PACKAGE

In questo package è presente la classe *WebAppUseCase*. Questa viene utilizzata per effettuare le richieste HTTP verso il *Service Layer*. Usufruisce dei metodi della classe *HTTPRequest* ed effettua le varie richieste GET, POST, PUT e DELETE a seconda del caso.

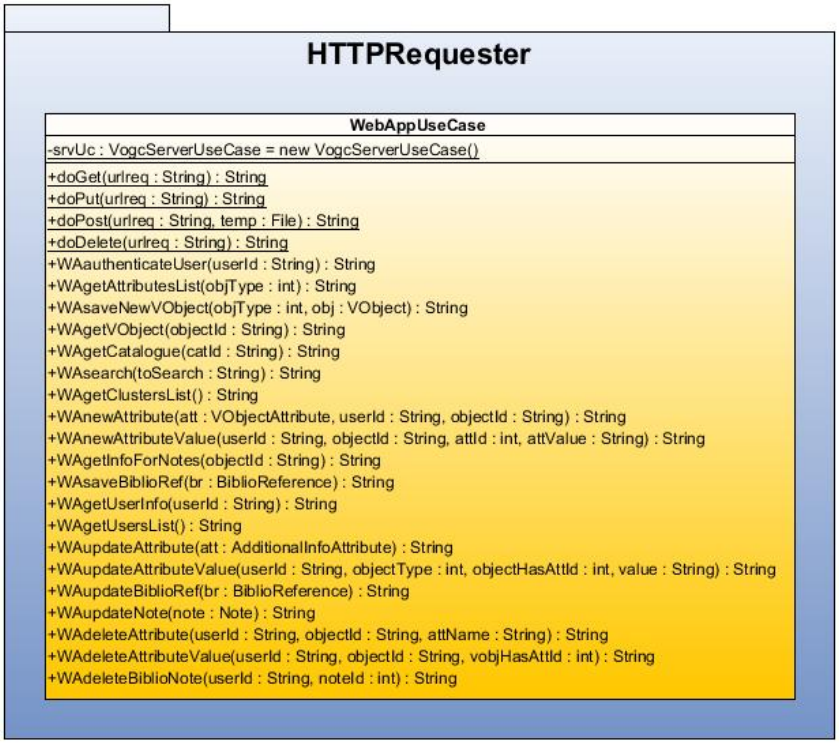

Figura 37: HTTPRequester Package

3.4.1.3 SERVICEDECLARATION PACKAGE

In questo package sono presenti le interfacce sincrone con le dichiarazioni dei metodi RPC. Ogni interfaccia sincrona, implementa *RemoteService* e descrive un particolare "servizio". Per ognuna di essa è presente la corrispondente interfaccia asincrona. Per una migliore comprensione tutte le interfacce sono state suddivise in base al servizio offerto.

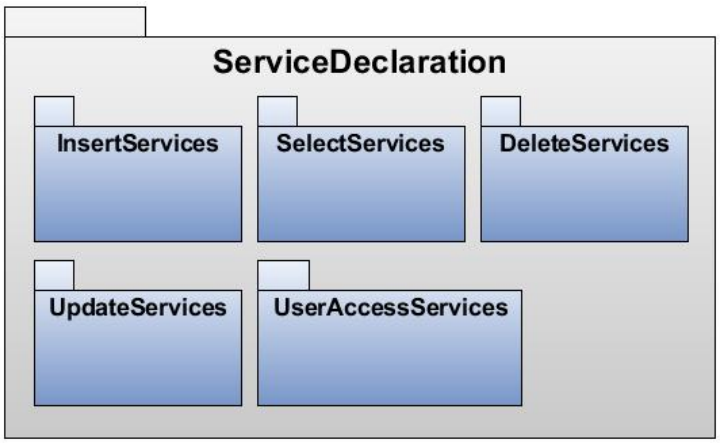

Figura 38: ServiceDeclaration Package

3.4.1.3.1 INSERTSERVICES PACKAGE

In questo package sono presenti tutte le interfacce che espongono servizi per l'inserimento di nuove informazioni.

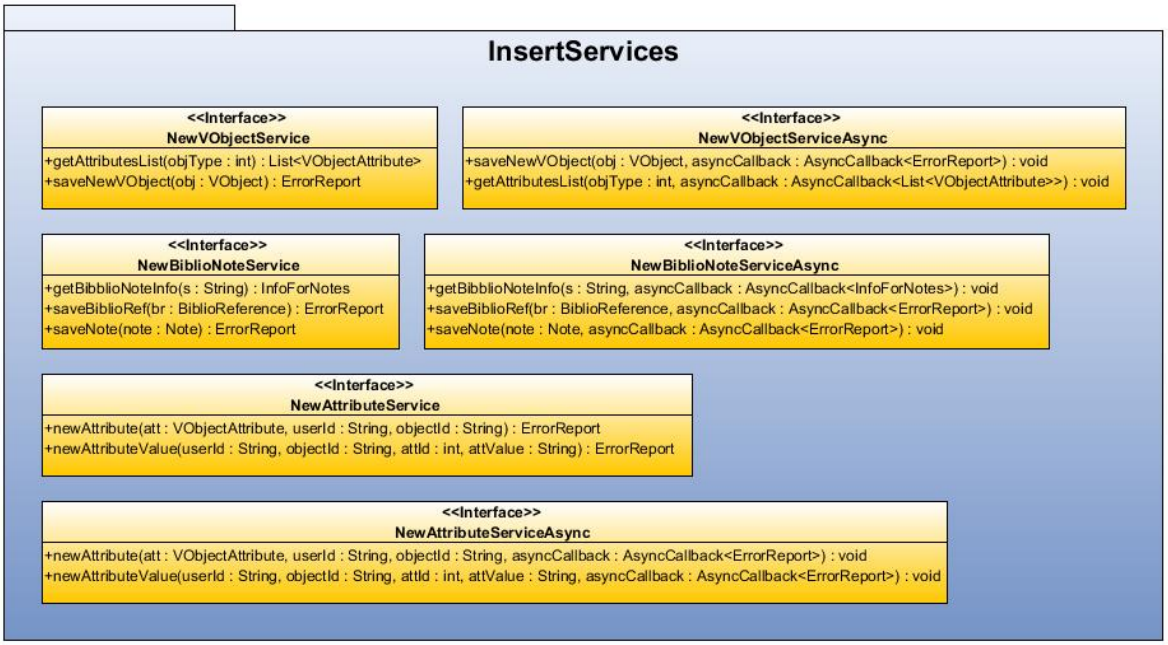

Figura 39: InsertServices Package

- **Inserimento nuovo oggetto (*NewVObjectService* – *NewVObjectServiceAsync*)**: queste interfacce contengono i seguenti metodi: *getAttributelist* e *saveNewVObject*. Il primo viene utilizzato per recuperare la lista aggiornata degli attributi e il secondo per il salvataggio del nuovo oggetto (ammasso, pulsar o stella).

- **Inserimento nuova nota bibliografica (*NewBiblioNoteService* – *NewBiblioNoteServiceAsync*)**: queste interfacce contengono i seguenti metodi: *getBiblioNotesInfo*, *saveBiblioRef* e *saveNote*. Il primo viene utilizzato per recuperare la lista aggiornata delle riviste e degli autori utile in fase di inserimento di una nuova nota. Mentre gli altri due vengono utilizzati per il salvataggio di un nuovo riferimento bibliografico e per il salvataggio di una nuova nota.

- **Inserimento nuovo attributo (*NewAttributeService - NewAttributeServiceAsync*):** queste interfacce contengono i seguenti metodi: *newAttribute* e *newAttributeValue*. Il primo viene utilizzato per l'inserimento di un nuovo tipo di attributo mentre il secondo per l'inserimento di un nuovo valore per un attributo esistente.

3.4.1.3.2 SELECTSERVICES PACKAGE

In questo package sono presenti tutte le interfacce che espongono servizi per la selezione di informazioni.

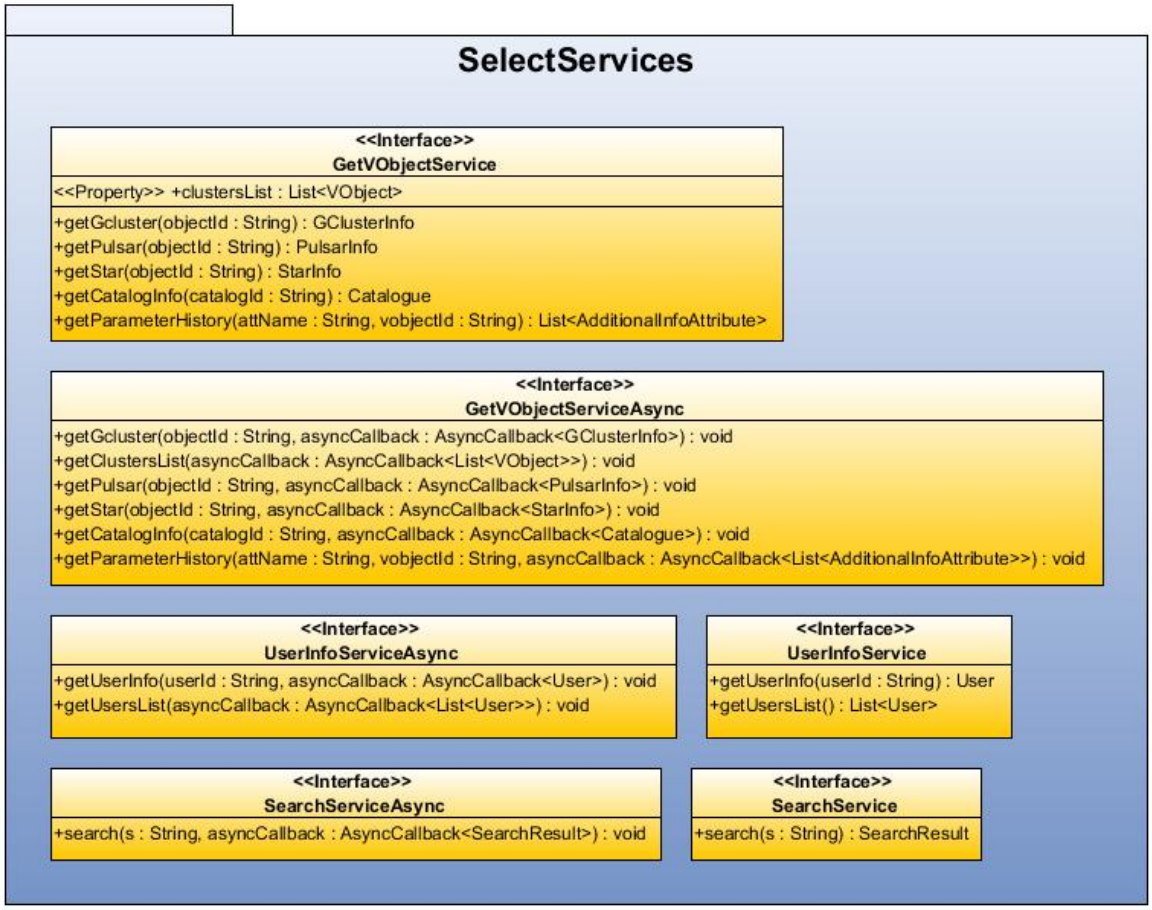

Figura 40: SelectServices Package

- **Visualizzazione delle informazioni relative ad un oggetto (*GetVObjectService* - *GetVObjectServiceAsync*):** in queste interfacce sono presenti i metodi: *getGcluster*, *getPulsar* e *getStar* utilizzati rispettivamente per la visualizzazione delle informazioni relative ad un ammasso globulare, ad una pulsar e ad una stella, poi i metodi *getCatalogueInfo* e *getParameterHistory* utilizzati per la visualizzazione delle informazioni relative ad un catalogo e per la visualizzazione della storia dei valori di un determinato parametro.

- **Visualizzazioni delle informazioni relative agli utenti (*UserInfoService* - *UserInfoServiceAsync*):** in queste interfacce sono presenti i metodi. getUserInfo e getUsersList. Il primo viene utilizzato per la visualizzazione delle informazioni relative ad

un singolo utenti, mentre il secondo viene utilizzato per la visualizzazione della lista degli utenti DAME abilitati.

- **Ricerca informazioni (*SearchService* - *SearchServiceAsync*):** in queste interfacce è presente il metodo *search* utilizzato per ricercare oggetti o immagini all'interno di VOGCLUSTERS.

Per una migliore comprensione, di seguito viene mostrato il codice delle interfacce *SearchService* e *SearchServiceAsync*:

```
1  import com.google.gwt.user.client.rpc.RemoteService;
2  import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;
3  import org.vogc.client.datatype.SearchResult;
4
5  /**
6   *
7   * @author Sabrina Checola
8   */
9  @RemoteServiceRelativePath("searchservice")
10 public interface SearchService extends RemoteService {
11     public SearchResult search(String s);
12 }
```

Figura 41: Interfaccia SearchService

```
1  import com.google.gwt.user.client.rpc.AsyncCallback;
2  import org.vogc.client.datatype.SearchResult;
3
4  /**
5   *
6   * @author Sabrina Checola
7   */
8  public interface SearchServiceAsync {
9
10     public void search(String s, AsyncCallback<SearchResult> asyncCallback);
11 }
```

Figura 42: Interfaccia SearchServiceAsync

3.4.1.3.3 UPDATESERVICES PACKAGE

In questo package sono presenti tutte le interfacce che espongono servizi per l'aggiornamento delle informazioni.

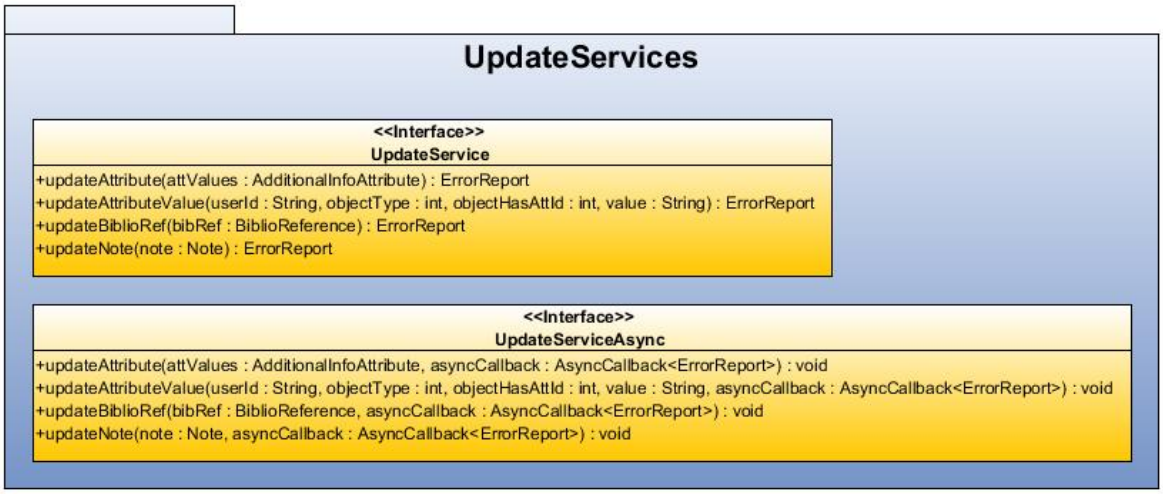

Figura 43: Package UpdateServices

- **Aggiornamento informazioni (*UpdateService* – *UpdateServiceAsync*)**: in queste interfacce sono presenti i metodi: *updateAttribute* utilizzato per aggiornare le informazioni descrittive di un attributo (escluso il valore assunto), *updateAttributeValue* utilizzato per aggiornare il valore di un attributo, *updateBiblioRef* utilizzato per aggiornare un riferimento bibliografico e infine *updateNote* utilizzato per aggiornare una nota.

3.4.1.3.4 DELETESERVICES PACKAGE

In questo package sono presenti tutte le interfacce che espongono servizi per la cancellazione delle informazioni.

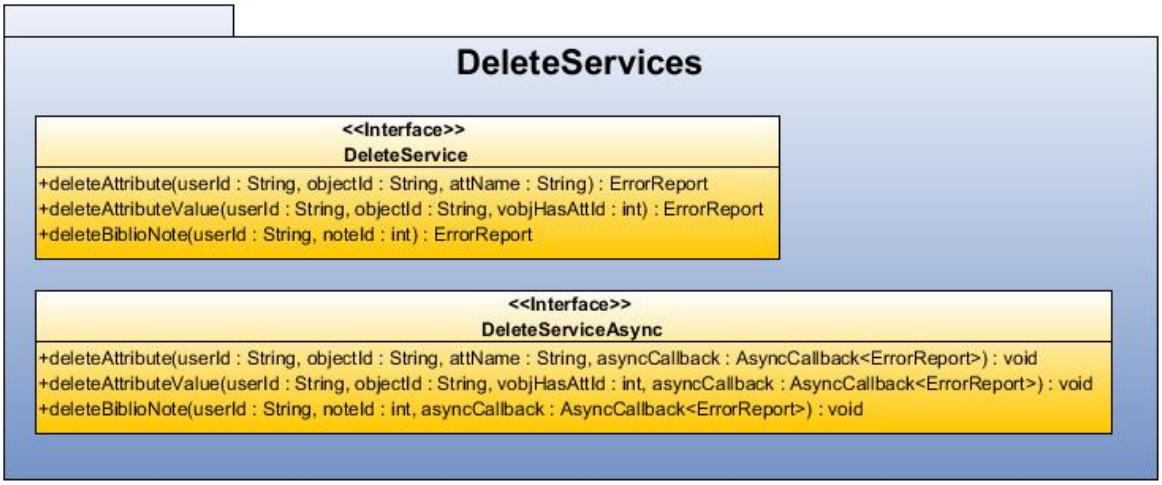

Figura 44: DeleteServices Package

- **Cancellazione informazioni (*DeleteService* – *DeleteServiceAsync*)**: in queste interfacce sono presenti i metodi: *deleteAttribute* utilizzato per cancellare tutte le informazioni relative ad un attributo, *deleteAttributeValue* utilizzato per cancellare un valore di un attributo, *deleteBiblioNote* utilizzato per cancellare un riferimento bibliografico.

3.4.1.3.5 USERACCESSSERVICES PACKAGE

In questo package sono presenti tutte le interfacce che espongono servizi relativi all'accesso e all'abilitazione degli utenti.

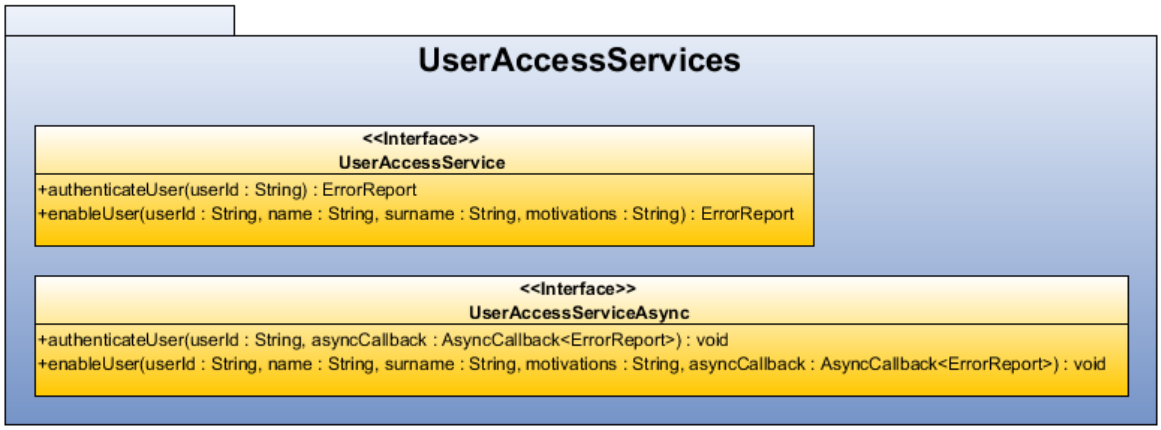

Figura 45: UserAccessService Package

- **Accesso/Abilitazione utente (*UserAccessService* – *UserAccessServiceAsync*)**: in queste interfacce sono presenti i metodi: *authenticateUser* utilizzato per autenticare un utente e *enableUser* utilizzato per attivare o disattivare l'account di un utente.

3.4.1.4 SERVICEIMPLEMENTATION PACKAGE

In questo package sono presenti le classi che estendono *RemoteServiceServlet* e che danno corpo ai metodi RPC dichiarati nelle interfacce sincrone descritte in precedenza.

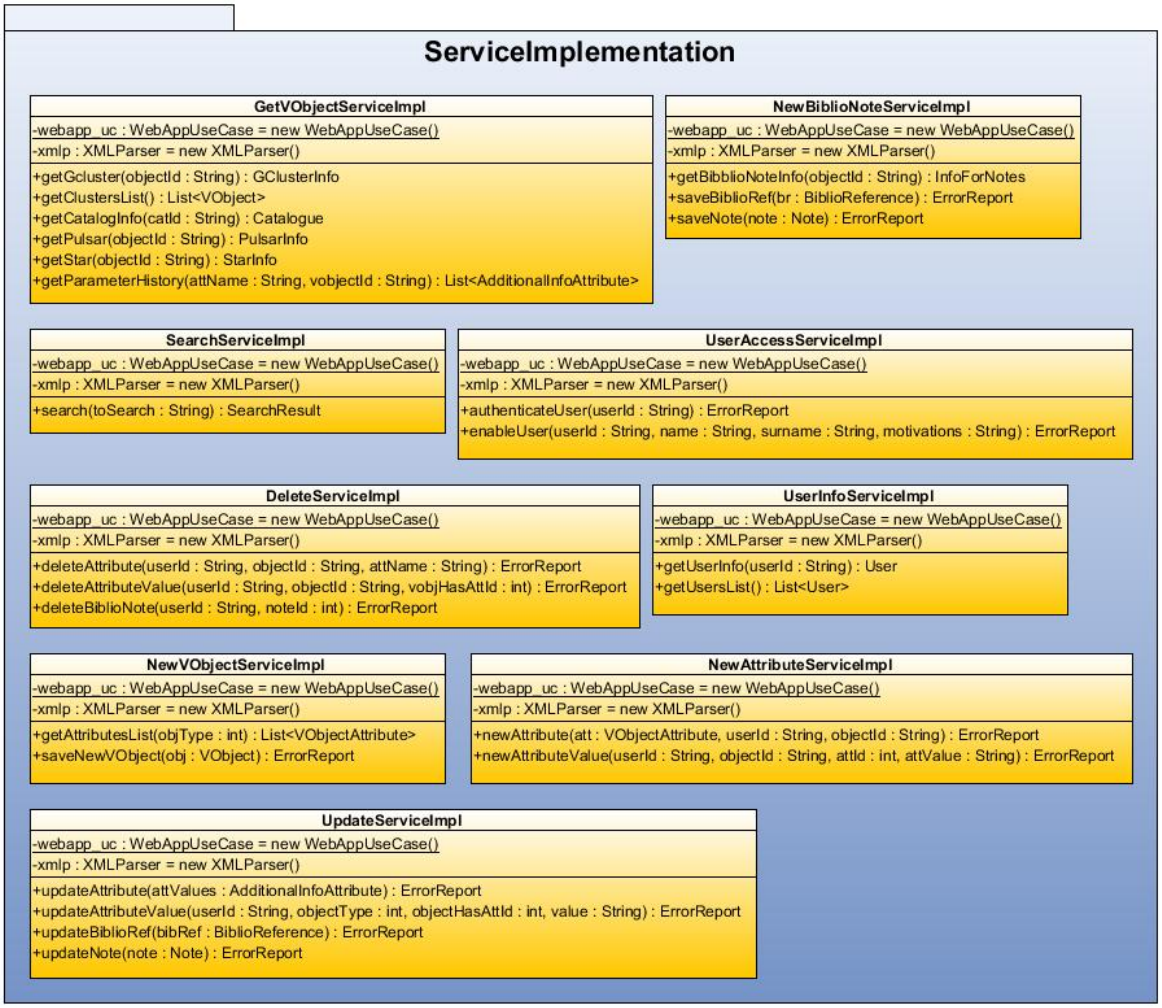

Figura 46: Package ServiceImplementation

- ***GetVObjectServiceImplementation*** implementa l'interfaccia *GetVobjectService*.
- ***NewBiblioNoteServiceImpl*** implementa l'interfaccia *NewBiblioNoteService*.
- ***SearchServiceImpl*** implementa l'interfaccia *SearchService*.
- ***UserAccessServiceImpl*** implementa l'interfaccia *UserAccessService*.
- ***DeleteServiceImpl*** implementa l'interfaccia *DeleteService*.
- ***UserInfoServiceImpl*** implementa l'interfaccia *UserInfoService*.

- ***NewVObjectServiceImpl*** implementa l'interfaccia *NewVObjectService*.
- ***NewAttributeServiceImpl*** implementa l'interfaccia *NewAttributeService*.
- ***UpdateServiceImpl*** implementa l'interfaccia *UpdateService*.

Anche in questo caso viene mostrato il codice della classe che implementa il servizio di ricerca:

```
1  import com.google.gwt.user.server.rpc.RemoteServiceServlet;
2  import freemarker.template.TemplateException;
3  import java.util.logging.Level;
4  import java.util.logging.Logger;
5  import org.vogc.client.SearchService;
6  import org.vogc.client.datatype.ErrorReport;
7  import org.vogc.client.datatype.SearchResult;
8  import org.vogc.server.LowLevel.Messages;
9  import org.vogc.server.LowLevel.WebAppUseCase;
10
11 /**
12  *
13  * @author Sabrina Checola
14  */
15 public class SearchServiceImpl extends RemoteServiceServlet implements SearchService {
16     private static WebAppUseCase webapp_uc = new WebAppUseCase();
17     private XMLParser xmlp = new XMLParser();   costruzione del parser XML
18
19     public SearchResult search(String toSearch) {
20         SearchResult src = new SearchResult();
21         try {
22             ErrorReport report = new ErrorReport();
23             String result = webapp_uc.WAsearch(toSearch);   chiamata alla classe che si occupa di effettuare le richieste HTTP al server.
24             if (result.equals(Messages.CONNECTION_NOT_FOUND)) {
25                 report.setCode(Messages.CONNECTION_NOT_FOUND);
26                 report.setMessage(Messages.getErrorString(Messages.CONNECTION_NOT_FOUND));
27                 report.setAdditionalInfo("");
28                 src.setReport(report);
29             } else {
30                 src = xmlp.parseSearchResult(result);   utilizzo del parser XML per analizzare il risultato restituito dal server
31             }
32         } catch (TemplateException ex) {
33             Logger.getLogger(SearchServiceImpl.class.getName()).log(Level.SEVERE, null, ex);
34         }
35         return src;   Oggetto da restituire alla GUI contenente i risultati della ricerca
36     }
37 }
```

Figura 47: Classe SearchServiceImpl

3.4.1.5 XMLGENERATORANDPARSER PACKAGE

Analogo al package descritto nel paragrafo 3.3.1.2, contiene due classi che però questa volta effettuano *parsing* dei documenti ricevuti dal Server e la generazione dei documenti da inviare al Server.

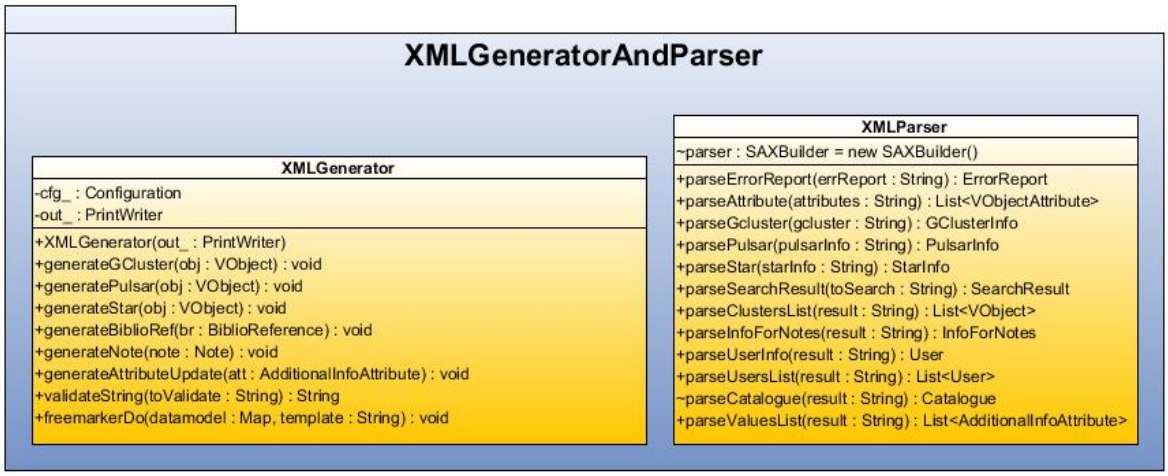

Figura 48: XMLGeneratorAndParser Package (FrontEnd Layer)

Anche in questo caso sono state utilizzate le API di JDOM per il *parsing* (classe *XMLParser*) e le API di Freemarker per la generazione (classe *XMLGenerator*).

3.4.1.6 GUI PACKAGE

In questo package sono presenti le classi che si occupano della visualizzazione delle informazioni. Per una questione di spazio, si è scelto di omettere il diagramma delle classi che non sarebbe stato di facile comprensione. Lo scopo di questo paragrafo infatti consiste nell'illustrare quali siano stati i passi fondamentali nella costruzione delle classi che formano l'interfaccia grafica usufruendo degli strumenti standard messi a disposizione da GWT.

Al fine di riprendere gli esempi mostrati nei paragrafi precedenti, di seguito è riportata una porzione di codice della classe che si occupa di offrire strumenti per effettuare la ricerca di oggetti all'interno di VOGCLUSTERS.

Questa classe estende *VerticalPanel*, uno dei *widget* grafici di GWT. Rappresenta quindi un pannello che contiene al suo interno una *textbox* per l'immissione della stringa da ricercare e un *button* per effettuare la chiamata asincrona. Il pannello e gli oggetti in esso contenuti vengono inglobati in un pannello principale.

La ricerca viene invocata attraverso il click del bottone da parte dell'utente. L'evento del click scatenerà la chiamata asincrona attraverso la quale verranno invocati i servizi esposti dalla classe *SearchServiceAsync* e implementati dalla classe *SearchServiceImpl*.

```
1  public class SearchServicePanel extends VerticalPanel {
2      private Label lblServerReply = new Label();
3      private TextBox txtUserInput = new TextBox();
4      private Button btnSend = new Button("SEARCH");
5      private Grid myGrid = new Grid(100, 100);
6
7      public SearchServicePanel() {
8          add(new Label("Text to Search: "));
9          add(txtUserInput);
10         add(btnSend);
11         add(lblServerReply);
12         add(myGrid);
13
14         // Create an asynchronous callback to handle the result.
15         Final AsyncCallback<SearchResult> callback = new AsyncCallback<SearchResult>() {
16             public void onSuccess(SearchResult result) {
17
18             if(result.getReport() == null){
19                 lblServerReply.setText("Search Result: ");
20                 Iterator nrow = result.getImages().iterator();
21                 int row = 0;
22                 while(nrow.hasNext()){
23                     Image img = (Image) nrow.next();
24                     myGrid.setText(row, 0, "" + img.getName());
25                     row++;
26                 }
27                 }else{
28                     lblServerReply.setText("Search Result: "+result.getReport().getMessage());
29                 }
30             }
31
32             public void onFailure(Throwable caught) {
33                 lblServerReply.setText("Communication failed");
34             }
35         };
36
37         // Listen for the button clicks
38         btnSend.addClickHandler(new ClickHandler(){
39             public void onClick(ClickEvent event) {
40                 // Make remote call. Control flow will continue immediately and later
41                 // 'callback' will be invoked when the RPC completes.
42                 getService().search(txtUserInput.getText(), callback);
43             }
44         });
45     }
46
47     public static SearchServiceAsync getService() {
48         // Create the client proxy. Note that although you are creating the
49         // service interface proper, you cast the result to the asynchronous
50         // version of the interface. The cast is always safe because the
51         // generated proxy implements the asynchronous interface automatically.
52
53         return GWT.create(SearchService.class);
54     }
55 }
56
```

Aggiunge il pannello di ricerca e gli oggetti ad esso annessi al pannello principale

questo oggetto permette di ricevere una risposta differita nel tempo da parte del back end. Verrà passato al metodo RPC

Se la risposta dal server è positiva si esegue aggiornamento della tabella con i valori relativi alla ricerca. Il valore ritornato viene inserito all'interno dell' oggetto di callback

qui viene gestire l'errore con un messaggio da mostrare all'utente

Aggiunge la gestione dell'evento click sul pulsante "cerca"

il metodo di callback RPC riceve i parametri seguiti obbligatoriamente dall' oggetto callback che verrà usato per la risposta asincrona

Figura 49: Codice pannello ricerca

Il codice evidenzia i passaggi fondamentali per eseguire l'invocazione del servizio e popolare una tabella con il risultato. In questo caso viene mostrato un esempio di come avviene la chiamata al servizio offerto dall'interfaccia *SearchService*.

È possibile quindi comprendere al meglio quello di cui si era discusso nel Capitolo 2. Si noti come la comunicazione avvenga tramite l'invocazione di una RPC ovvero tramite invocazione asincrona di un oggetto remoto. La GUI, dopo aver invocato il server non attende la risposta ma ritorna subito alla interattività con l'utente. La risposta viene assegnata ad un oggetto *AsyncCallback* che ingloba al suo interno l'oggetto restituito dal server (nell'esempio è l'oggetto *SearchResult*).

GWT viene rilasciato con un set minimo di componenti grafici (*widget*) tramite i quali è stato possibile realizzare un'interfaccia grafica interattiva (caratteristica tipica di una applicazione Ajax), ma piuttosto semplice con una grafica non molto accattivante.

Questo è uno dei motivi che condurrà ad una nuova release della web application con una GUI differente realizzata con una delle nuove librerie di widget.

Un fattore piuttosto interessante legato a GWT è la presenza di una community molto attiva che sta rilasciando in modo open source, gratuitamente o a pagamento, alcuni progetti che si aggiungono a quanto proposto da Google. In tal senso molto interessanti sono i vari framework che si vanno a sovrapporre al kit base e che permettono di usare set di componenti molto versatili e potenti.

Questo aspetto verrà discusso nel capitolo finale di questa tesi.

Capitolo 4

Testing

4.1 TECNICHE DI COLLAUDO DEL SOFTWARE

Il testing rappresenta una delle attività più importanti per assicurare la qualità del software. Dopo aver generato il codice sorgente, si deve collaudare il software per scoprire (e correggere) quanti più errori possibili prima di fornire il prodotto al cliente. Lo scopo è quello di progettare una serie di casi di collaudo che abbiano un'elevata probabilità di individuare gli errori.

Ogni prodotto può essere collaudato in due modi:

- **Testing *black-box***: approccio utilizzato quando si conoscono le funzioni specifiche che il prodotto deve svolgere. In questo caso si possono effettuare prove tese a dimostrare che tali funzioni sono effettivamente svolte, cercando nel contempo eventuali errori in ciascuna funzione;

- **Testing *white-box***: approccio utilizzato quando è nota la struttura interna del prodotto. In questo caso si possono effettuare prove tese a garantire che le operazioni interne rispettino le specifiche, avendo cura di mettere alla prova tutti i componenti.

Quindi in definitiva, il testing black-box tende a dimostrare che il software svolge correttamente le sue funzioni, senza preoccuparsi della struttura logica interna. Il testing white-box, invece si basa su un esame meticoloso degli aspetti procedurali.

La strategia utilizzata per testare le componenti di VOGCLUSTERS è il testing black-box. Ogni componente, una volta ultimata, è stata sottoposta a collaudo. Di seguito verrà mostrato un test case di esempio con il relativo codice.

4.2 TEST CASE

Come detto, la tecnica *black-box* è stata adottata per verificare, mediante pianificazione ed esecuzione di vari test case specifici, la correttezza sintattica e funzionale delle varie operazioni previste dall'applicazione, sempre con particolare riferimento al flusso I/O tra le varie componenti. In questo paragrafo, a titolo esemplificativo e per brevità, viene illustrato uno di questi test case

relativi alle funzionalità. Quest'ultima, nel caso specifico di questo esempio, è l'autenticazione dell'utente. Di seguito verrà mostrato il caso di test della servlet che implementa tale funzionalità.

Servlet URL: `GET/SERVER/authenticate?userId=x`

input: username

output: messaggio di successo con tipologia di utente oppure un messaggio di errore

Test Case Number	Description	Expected Result	Effective Result	Type
1/Enabled user, user admin DAME	admin	User Admin DAME Enabled	OK	AUTO
2/Enabled user, user DAME	utente1@inventata.it	User DAME enabled	OK	AUTO
3/User not enabled	utente2@inventata.it	Error: User DAME not enabled	OK	AUTO
4/Wrong userId	utente@	Error: wrong userId	OK	AUTO
5/Null userId	An empty string	Error: null userId	OK	AUTO

Figura 50: Testcase autenticazione utente

L'output di ogni test case è stato riportato in un file di testo. L'output del caso di test analizzato in questo esempio è il seguente:

```
test 1
<?xml version="1.0" ?>
<VOTABLE   version="1.1"   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1
http://www.ivoa.net/xml/VOTable/v1.1"
xmlns="http://www.ivoa.net/xml/VOTable/v1.1">
      <DESCRIPTION>Error Report</DESCRIPTION>
      <INFO name="errorCode" value="vogcSRV0002"/>
      <INFO name="errorMessage" value="User Admin DAME Enabled"/>
      <INFO name="additionalInfo" value=" "/>
</VOTABLE>

test 2
<?xml version="1.0" ?>
<VOTABLE   version="1.1"   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1
http://www.ivoa.net/xml/VOTable/v1.1"
xmlns="http://www.ivoa.net/xml/VOTable/v1.1">
      <DESCRIPTION>Error Report</DESCRIPTION>
      <INFO name="errorCode" value="vogcSRV0001"/>
      <INFO name="errorMessage" value="User DAME Enabled"/>
      <INFO name="additionalInfo" value=" "/>
</VOTABLE>

test 3
<?xml version="1.0" ?>
<VOTABLE   version="1.1"   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1
http://www.ivoa.net/xml/VOTable/v1.1"
xmlns="http://www.ivoa.net/xml/VOTable/v1.1">
       <DESCRIPTION>Error Report</DESCRIPTION>
       <INFO name="errorCode" value="vogcSRV0003"/>
       <INFO name="errorMessage" value="User DAME Not Enabled"/>
       <INFO name="additionalInfo" value=" "/>
</VOTABLE>

test 4
<?xml version="1.0" ?>
<VOTABLE   version="1.1"   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1
http://www.ivoa.net/xml/VOTable/v1.1"
xmlns="http://www.ivoa.net/xml/VOTable/v1.1">
       <DESCRIPTION>Error Report</DESCRIPTION>
       <INFO name="errorCode" value=" vogcDB0001"/>
       <INFO name="errorMessage" value=" Error: User Not Exists"/>
       <INFO name="additionalInfo" value=" "/>
</VOTABLE>

test 5
<?xml version="1.0" ?>
<VOTABLE   version="1.1"   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.1
http://www.ivoa.net/xml/VOTable/v1.1"
xmlns="http://www.ivoa.net/xml/VOTable/v1.1">
       <DESCRIPTION>Error Report</DESCRIPTION>
       <INFO name="errorCode" value=" vogcSRV0004"/>
       <INFO name="errorMessage" value="Error: UserId is Null"/>
       <INFO name="additionalInfo" value=" "/>
</VOTABLE>
```

Figura 51: Output Testcase

4.3 RISULTATI DEL TESTING

La più volte menzionata strategia di progettazione ed implementazione di VOGCLUSTERS, basata su componenti, trova nella fase del test e collaudo finale la massima conferma dei suoi vantaggi. La scomposizione dei *test case* funzionali complessi in casi più semplici e localizzati tra due singoli componenti o sub-componenti sicuramente ne facilita l'esecuzione, la comprensione e soprattutto l'intervento in caso di *bug* o errato comportamento.

I *test case* svolti in VOGCLUSTERS hanno permesso di emulare l'utenza della web application secondo le diverse tipologie di accesso, ma non solo. Hanno permesso di verificare la correttezza delle procedure di inserimento, modifica e cancellazione di informazioni, nonché la validità e la correttezza delle procedure di *parsing* e generazione di documenti XML.

Per testare la validità delle procedure di accesso al db sono stati svolti diversi *test case* per ogni classe della componente VOGCACCESS. In questa fase la percentuale di errore è stata di circa 15%.

Per testare la validità delle procedure di accesso alle funzionalità del server (componente SERVER) sono stati svolti *test case* per ogni classe presente nei package Web Service e XMLGeneratorAndParser. In questo caso la percentuale di errore è stata circa del 30%.

Infine per testare tutta la logica relativa al frontend, sono state testate le RemoteServiceServlet che hanno consentito di verificare la correttezza di ogni singolo metodo RPC e di ogni procedura di parsing e generazione di XML. Anche in questo caso la percentuale di errori riscontrati è stata del 30% circa.

Nelle successive release di VOGCLUSTERS, per un collaudo più accurato, è prevista la possibilità di adottare anche la metodologia *white-box*.

Capitolo 5

Sviluppi futuri

Come accennato nel paragrafo GUI PACKAGE, uno dei propositi per il futuro, è quello di realizzare un'interfaccia grafica più interattiva. I motivi che mi hanno portato a questa considerazione sono due: (1) nello sviluppare l'interfaccia grafica con GWT mi sono resa conto che la libreria di componenti di base non era sufficiente per realizzare una GUI moderna e dinamica. Attualmente esistono librerie di widget (prima fra tutte SmartGWT[19]) che si integrano perfettamente in GWT e che mettono a disposizione un gran numero di funzionalità aggiuntive. Parte di queste funzionalità, nel mio caso è stato necessario realizzarle "manualmente", con notevole dispendio di tempo. (2) Attualmente sta per essere rilasciata una nuova release della suite DAME, la cui GUI è stata realizzata proprio con SmartGWT. Quindi, per una questione di uniformità, sia a livello grafico, sia a livello di funzionalità offerte, sarà necessario apportare modifiche alla GUI di VOGCLUSTERS, includendo i nuovi widget di SmartGWT. Questo ovviamente non comporterà alcuna modifica al resto della suite.

Conclusioni

La web application VOGCLUSTERS è un progetto integrato nel Programma DAME, in collaborazione con il dott. M. Castellani dell'INAF OAR (Osservatorio Astronomico di Roma), il cui scopo principale è la realizzazione di una Framework Suite di strumenti per data mining ed esplorazione di massive data set, basata su strumenti web 2.0 e piattaforme di calcolo distribuito. In tale contesto, del tutto volutamente generale e trasversale rispetto a qualsiasi disciplina scientifica o tecnologica, VOGCLUSTERS condivide molte peculiarità strutturali e standard progettuali di DAME. La sua originalità è inoltre nell'elevato grado di specializzazione applicativa, nonostante mantenga perfettamente integro il carattere multidisciplinare dell'intero Programma DAME. Infatti, l'obiettivo primario del progetto VOGCLUSTERS è la realizzazione di un framework DAME-based per l'esplorazione e mining di archivi di dati relativi ad ammassi globulari, una peculiare tipologia di oggetti astronomici, che richiede strumenti di cross-correlation non solo analitici o statistici, ma anche basati su ricerche complesse di tipo bibliografico e inferenziale, richiedendo quindi l'impiego di regole e strumenti ipertestuali. Tali caratteristiche, unite agli indubbi vantaggi di essere basato sullo stato dell'arte dell'ICT (tecnologie web 2.0, calcolo distribuito, etc.) e di impiegare gli strumenti standard del circuito IVOA per l'accesso condiviso ad archivi scientifici distribuiti, rendono il progetto non soltanto originale ed unico nel contesto disciplinare astrofisico per gli ammassi globulari, ma ne evidenziano di conseguenza anche la principale valenza in ambito puramente scientifico. L'elevata velocità e capacità di manipolazione delle informazioni, information retrieval, cross-correlation, pattern matching e soprattutto di interoperabilità tra archivi delocalizzati di dati astronomici osservati e archivi bibliografici e ipertestuali, costituiscono uno strumento a disposizione della comunità astrofisica, potenzialmente in grado di migliorare tempi e risultati della speculazione scientifica sugli ammassi globulari, favorire la multidisciplinarità mediante la disseminazione di strumenti ICT all'interno della comunità astronomica e stimolare l'aggregazione di gruppi di ricerca mediante l'uso di uno strumento comune di ricerca.

Ringraziamenti

Al dott. Massimo Brescia e ad Alfonso Nocella per tutto il supporto e l'aiuto fornito durante l'intero periodo di tirocinio.

Alla dott.ssa Anna Corazza per la disponibilità e i tanti consigli forniti durante la stesura della tesi.

Al dott. Marco Castellani, partnership ufficiale all'interno del progetto, per il supporto, la disponibilità e la consulenza scientifica forniti durante l'intera esecuzione del lavoro.

Bibliografia

Pressman, Roger S. *Principi di Ingegneria del software.* McGraw-Hill.

Bass, Clements, Katzman. *Software Architecture in Practice.* Addison-Wesley, 2003.

Cay S. Horstmann, Gary Cornell. *Core Java 2, Volume.* Pearson - Prentice Hall.

Diotalevi, Filippo. *Java Enterprice Ediction 5.* HOEPLI, 2006.

IvoaVOTable. http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/IvoaVOTable .

Java . http://java.sun.com.

MokaByte. *Manuale pratico di Java.* HOPS.

Progetto S.C.O.P.E. http://www.scope.unina.it.

Progetto Vo-Neural \ Dame. http://voneural.na.infn.it.

Shepherd, Devan. *XML Guida completa.* APOGEO.