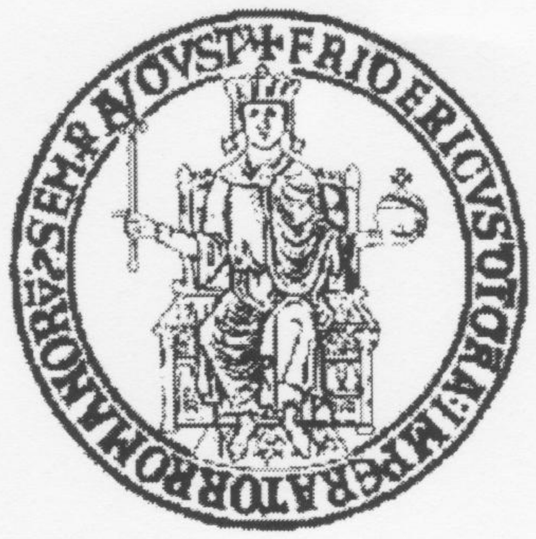
Università degli studi di Napoli Federico II
Facoltà di Scienze Naturali, Matematiche e Fisiche
Corso di Laurea in Informatica



Tesi di Laurea

Il componente Framework del progetto VO-Neural\Dame

Tutor accademico: Anna Corazza
Tutor aziendale: Massimo Brescia

Candidato: Michelangelo Fiore
Matricola: 566/2325

Anno accademico 2007-2008

Sommario

Indice delle figure

1. Introduzione

Il progetto VO-Neural \ DaME (Virtual Observatory-Neural \ Data Mining Exploration) nasce con lo scopo di fornire una suite di data mining orientata al web, che permette ad un utente di effettuare esperimenti di data mining astronomico di una certa complessità. La suite utilizza una potente infrastruttura computazionale, realizzata nel progetto SCoPE (Sistema Cooperativo per Esperimenti Scientifici ad alte prestazioni) dell'Università degli Studi di Napoli Federico II. Si tratta di una grande infrastruttura hardware, basata sul paradigma Grid, che unisce dipartimenti di Napoli dislocati in diverse zone della città.[1] Il progetto è stato curato da diversi membri ed è finanziato dall'Unione Europea e dal MIUR.

Partnership:

- Dipartimento di Fisica (sezione di 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
- Dipartimento di Informatica - Università degli Studi di Napoli Federico II
- Dipartimento di Ingegneria Informatica - Università degli Studi di Napoli Federico II
- MIUR (Ministero dell'Istruzione, dell'Università e della Ricerca)
- EURO-VO (Il laboratorio di osservazione virtuale europeo)

[1] http://www.scope.unina.it

Nell'ambito di questo progetto mi sono occupato, sotto la supervisione di Omar Laurino, prevalentemente della realizzazione del Framework, ovvero il componente che si occupa di coordinare il resto della suite in modo da completare le operazioni. A causa della natura del Framework mi sono trovato, però, in stretto contatto con gli altri membri del team ed in particolare ho partecipato al lavoro di ingegnerizzazione dei Data Mining Models, ovvero gli algoritmi utilizzati per gli esperimenti di data mining.

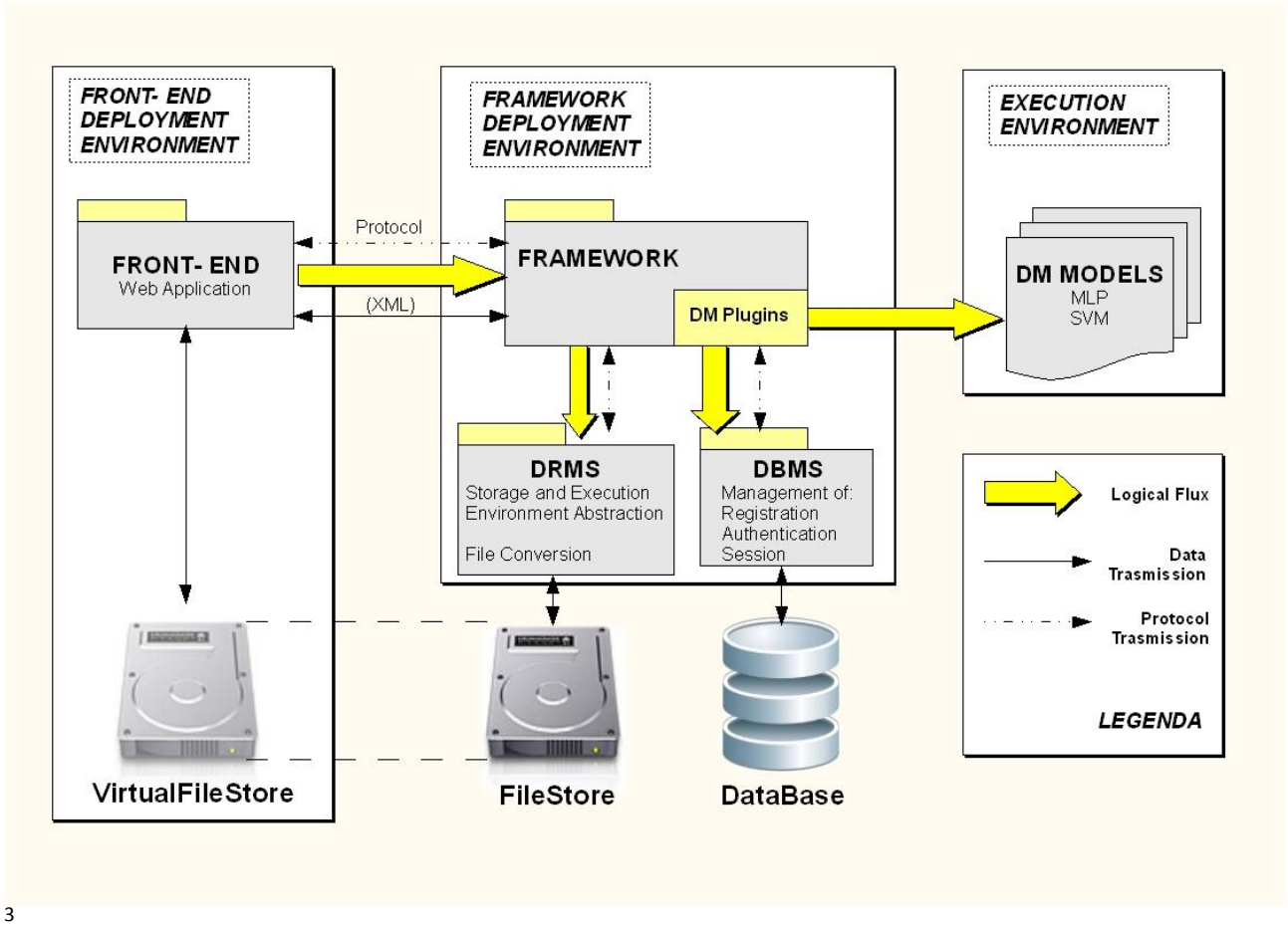
Il data mining, ovvero l'utilizzo di metodologie computazionali per estrarre informazioni da insiemi di dati, è un processo che ha acquisito una notevole importanza negli ultimi anni. Le sue applicazioni sono molto numerose e vanno dalla ricerca scientifica a operazioni di marketing. Generalmente gli insiemi di dati utilizzati risultano avere dimensioni molto elevate, quindi un algoritmo di data mining può impiegare un grosso tempo computazionale. Il data mining astronomico, in particolare, si basa su insiemi di dati estremamente grandi. Questo, unito all'alto costo computazionale degli algoritmi utilizzati, pone grossi limiti a questi problemi.

L'utilizzo di Grid permette alla suite di supportare gli esperimenti di un grande numero di utenti che, utilizzando una semplice interfaccia web, possono gestire i propri insiemi di dati, creare nuovi esperimenti e tenere traccia dei loro sviluppi. E' così possibile ottenere dati di output parziali durante l'esperimento, sotto forma di grafici, e finali al suo termine. Oltre a GRID, la suite è progettata per consentire esperimenti anche su normali computer Stand-Alone. Questo permetterà di smistare gli esperimenti su GRID o su macchine Stand-Alone a seconda di determinati parametri, come ad esempio la durata dell'esperimento.[2]

[2] Su questo vedi http://voneural.na.infn.it,

2. Il progetto VO-Neural \ DaME

2.1. Introduzione



[3]

Figura 1: Diagramma dei componenti

[3] M. Garofalo, O. Laurino. Vedi http://voneural.na.infn.it/project.html

A causa della complessità della suite, si è reso necessario scomporre l'applicazione in diversi componenti, affidati a diversi membri del team di sviluppo, che interagiscono strettamente per completare le operazioni. Escludendo il Front-End, ogni altro componente è affidato a due o più membri del team. Questa collaborazione è nata dalla necessità, dovuta alla complessità del progetto, di unire le diverse conoscenze del gruppo, che comprende informatici, fisici ed ingegneri.

I componenti della suite sono:

- Front-End : una web-application responsabile di gestire le interazioni con l'utente.
- Database Management System (DBMS): una libreria di classi, responsabile della gestione delle interazioni con il Database
- Driver Management System (DRMS): una libreria di classi che ha il compito di astrarre il Framework dalle interazioni con il file store e con l'ambiente di deployment.
- Framework: il cuore dell'applicazione, con il compito di gestire e coordinare tutti gli altri componenti.
- Data Mining Models (DMM): le librerie software che implementano algoritmi di data mining.

2.2. Il Front-End

Figura 2: Front-End

Realizzato da Francesco Manna, il Front-End è il componente responsabile di gestire le interazioni con l'utente. La suite di data-mining è pensata anche per utenti che non possiedano forti conoscenze informatiche. Lo scopo del Front-End è, quindi, realizzare una web-application con una semplice Graphical User Interface . Prima di poter accedere alle funzioni di data-mining, un utente deve registrarsi creando un account. A ogni visita successiva, l'utente dovrà effettuare il log-in per poter accedere alle altre funzionalità del sistema.

Un utente ha la possibilità di creare varie sessioni, che rappresentano le sue aree di lavoro. In ogni sessione, l'utente potrà creare esperimenti ed eseguire l'upload dei propri file, che potrà poi utilizzare negli esperimenti. Per effettuare i download e gli upload dei file, l'utente ha a disposizione un Virtual File Store, che astrae il File Store Fisico, gestito da Framework e DRMS. In ogni esperimento, l'utente dovrà selezionare la funzionalità da utilizzare, che corrisponde a un caso d'uso scientifico, scegliere i dati di input e selezionare gli output ai quali è interessato. Gli altri componenti della suite si occuperanno di eseguire l'esperimento. Ad esempio: un utente potrebbe voler effettuare un esperimento di classificazione su un proprio insieme di dati, utilizzando il modello di reti neurali MultiLayer Perceptron (MLP). L'utente dovrà scegliere la funzionalità classificazione MLP, dare un nome all'esperimento e scegliere la modalità di lancio tra alcune

opzioni. Supponiamo che l'utente scelga la modalità "run". In questo caso, dovrà semplicemente indicare i file che contengono l'insieme di dati da classificare e la rete neurale da utilizzare per poi lanciare l'esperimento.

Dopo aver lanciato un esperimento, il Front-End comunica con il Framework per conoscere il suo stato e ottenere i file di output. Altre comunicazioni tra i due componenti riguardano la lista delle sessioni e dei file di un utente, la lista delle funzionalità e la descrizione di una funzionalità. Queste comunicazioni avvengono tramite eXtensible Markup Language (XML). (vedere 3.1.2 per maggiori dettagli).

2.3. Driver Management System

Il Framework della suite di data-mining è stato pensato per poter funzionare su due diverse piattaforme: Grid e Stand-Alone. E' risultato fondamentale quindi costruire uno strato che astragga il Framework dalla piattaforma corrente. Questo strato è il Driver Management System (DRMS).

Nella prima versione del software, a ogni Framework sarà associato uno e un solo driver. In futuro, però, entrambi i driver saranno presenti su ogni Framework e il DRMS avrà il compito di scegliere, in maniera intelligente, quale piattaforma sia più opportuno utilizzare, in base a parametri dipendenti dall'esperimento.

Il DRMS e i rispettivi driver sono realizzati da Gianluca d'Angelo e Mauro Garofalo. I compiti di un Driver sono:

- La gestione dei file fisici, ovvero il loro download, upload, copia ed eliminazione.
- La traduzione di un file in un diverso formato. Ogni funzionalità ha delle proprie necessità di formato, ma l'utente può gestire i propri dati nel formato da lui preferito, quindi può essere necessario effettuare delle traduzioni prima dell'esecuzione di un esperimento.

- L’esecuzione di un esperimento. Nel caso di un driver Stand-Alone l’esperimento viene eseguito sulla macchina corrente e non sono necessarie operazioni complicate.
  L’infrastruttura Grid è però composta da un grosso numero di computer, chiamati Worker Node. Il driver Grid si deve quindi occupare anche della migrazione dell’esperimento sul Worker Node scelto per la sua esecuzione. Il Framework avrà in seguito la possibilità di interrogare il Driver per conoscere lo stato di un esperimento.

2.4. Il Database Management System

Il Database Management System (DBMS) è una libreria di classi responsabile della gestione delle comunicazioni tra il Framework ed il Database. E' stata realizzata da Alfonso Nocella. Il Framework non conserva nessuna traccia delle interazioni con l'utente, quindi, il Database rappresenta la sua memoria, mantenendo traccia di utenti, sessioni, files, esperimenti e funzionalità. Nelle figure di seguito si trovano lo schema ER del Database ed il class diagram del DBMS.

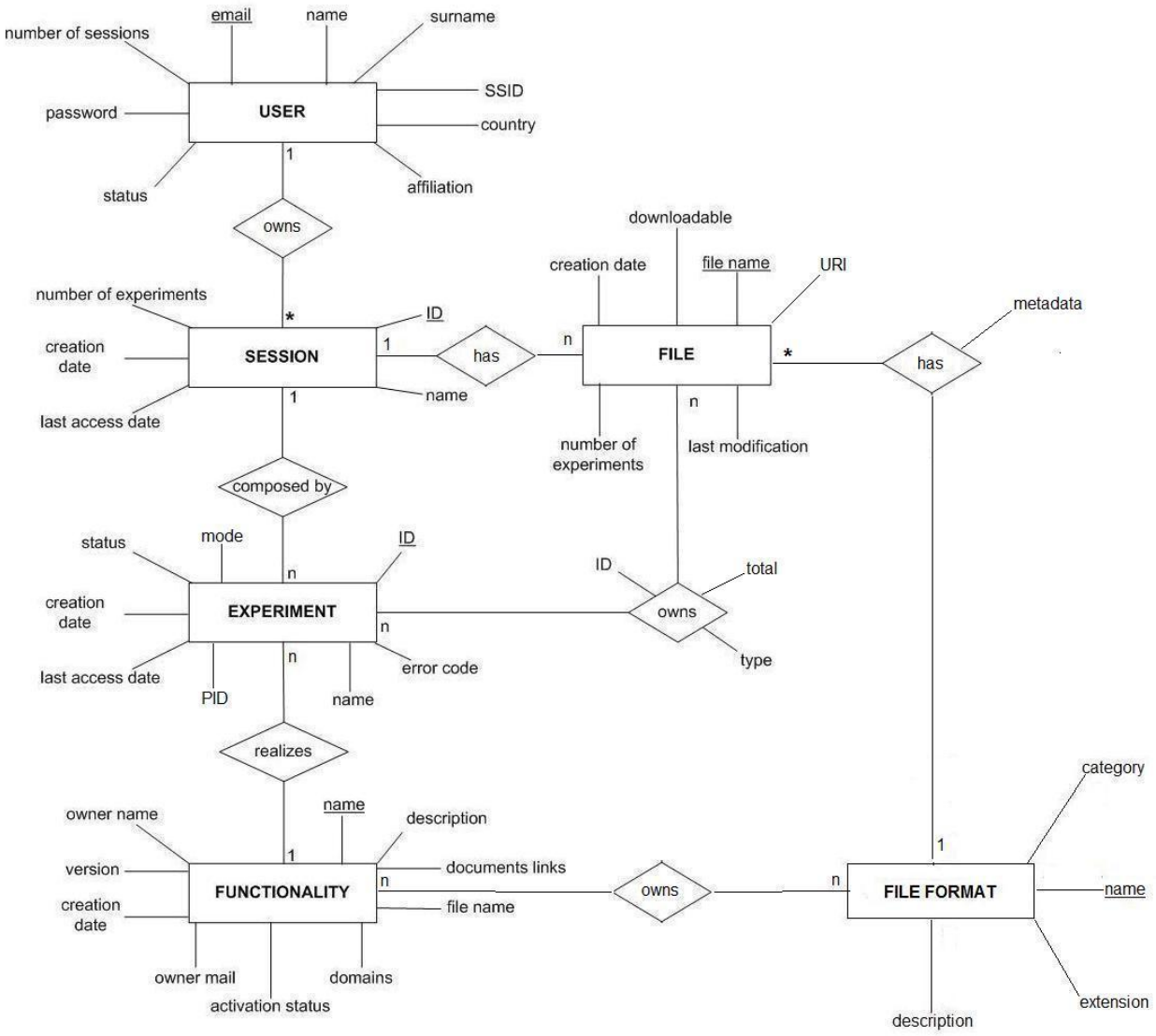


Figura 3: schema ER Database

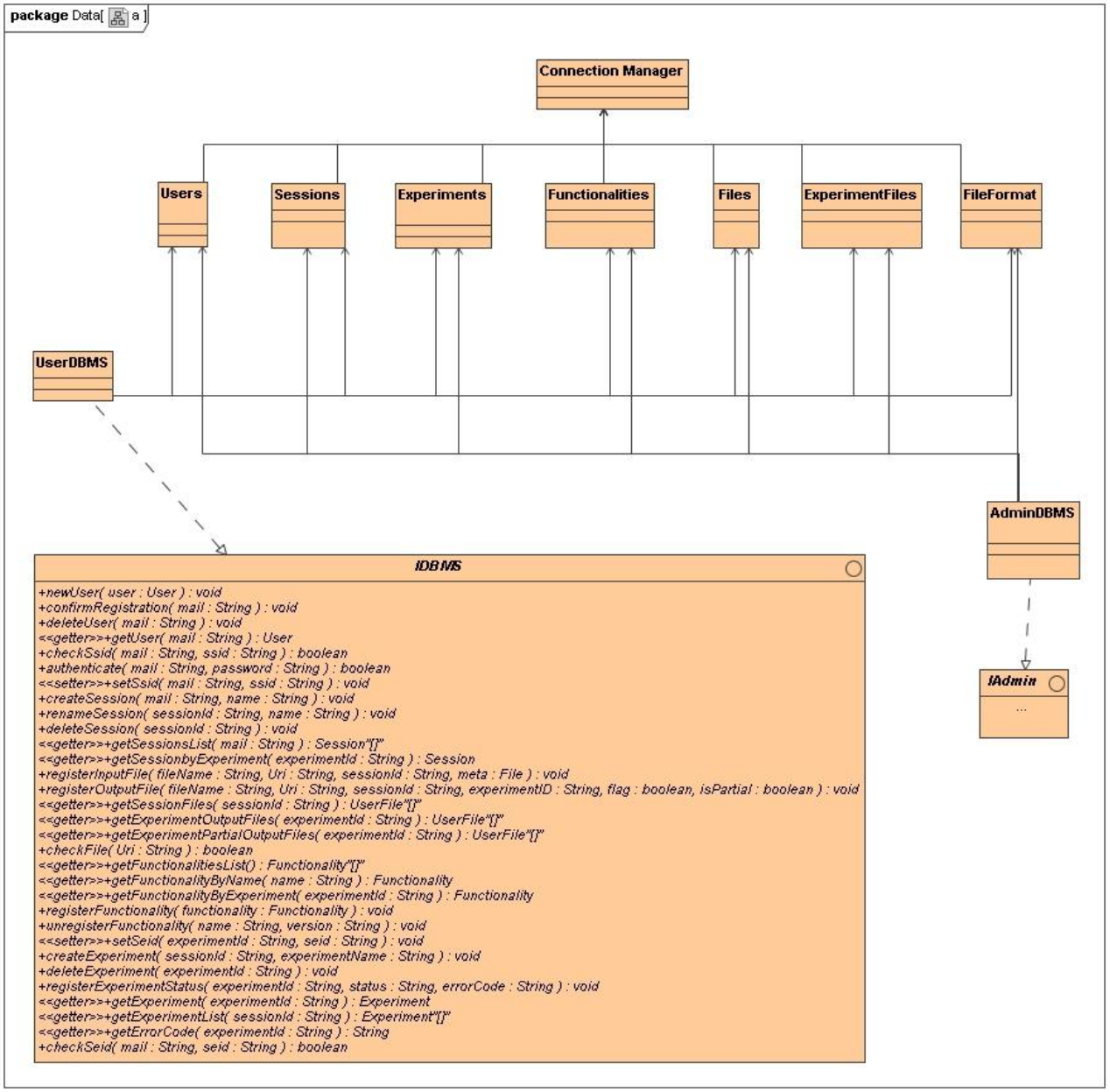


Figura 4: Class Diagram DBMS

2.5. I Data Mining Models

I Data Mining Models sono algoritmi utilizzati per operazioni di data mining. Ogni funzionalità del sistema deve utilizzare uno o più modelli per completare un esperimento. Il compito di Stefano Cavuoti, aiutato da me e da Omar Laurino, è quello di costruire una libreria di classi, in un ambiente Object-Oriented, che rappresenti questi modelli come gerarchia di classi, in modo da favorire il più possibile il riutilizzo del codice e del polimorfismo. Lo sviluppatore di una funzionalità dovrebbe limitarsi a instanziare i modelli necessari e a utilizzare i loro metodi, trattandoli come una scatola nera. I modelli potrebbero essere scritti da vari sviluppatori in diversi linguaggi, e potrebbero anche essere file eseguibili. Una parte fondamentale del lavoro di ingegnerizzazione consiste, quindi, nel creare dei wrapper attorno ai file eseguibili e nell'utilizzare chiamate native per i modelli scritti in linguaggi diversi da quello utilizzato per l'implementazione della suite. Questa gerarchia di classi deve partire da concetti estremamente astratti, come un classificatore, andando a dettagliarli il più possibile tramite specializzazioni. A causa della natura dei modelli, il lavoro di ingegnerizzazione ha incontrato alcuni problemi, sia nella definizione delle classi, sia nel rapporto tra queste. In particolare, una grossa percentuale del lavoro è stata spesa nel definire le relazioni che intercorrono tra un modello e una funzionalità. E' stato necessario analizzare svariate alternative prima di trovare la soluzione più vicina alle nostre necessità.

I Data Mining Models sono trattati con dettaglio nel capitolo 3.2

2.6. Il Framework

Realizzato da me e da Omar Laurino, il Framework è il nucleo della suite ed è responsabile del coordinamento degli altri componenti. I compiti del Framework sono:

- Rispondere alle richieste del Front-End, interagendo con DRMS e DBMS.
- Presentare un'interfaccia di amministrazione, che consenta ad un admin di installare nuove funzionalità ed effettuare operazioni statistiche sul sistema.

Il Framework è trattato dettagliatamente nel capitolo 3.

3. Il Framework

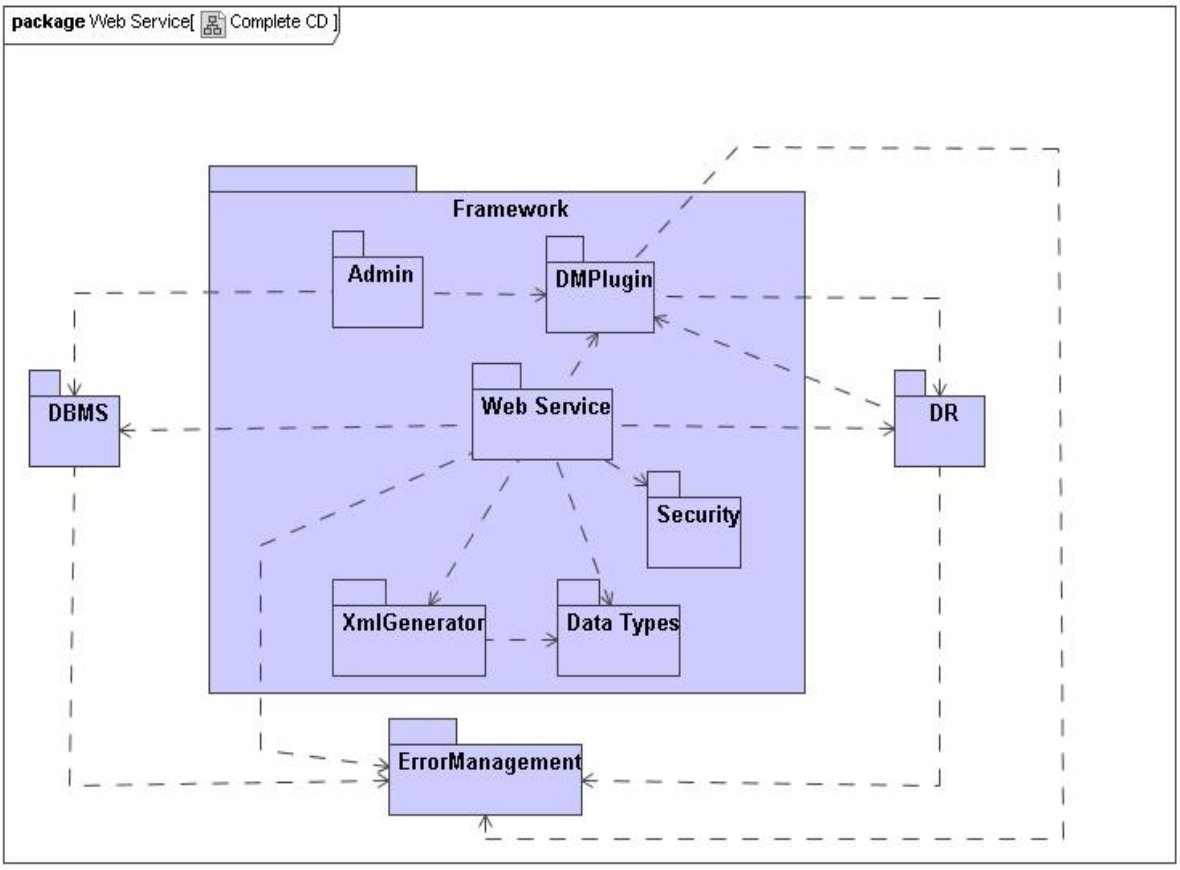


Figura 5: Framework Class Diagram

Durante l'analisi del Framework è emersa la necessità di dividerlo in tre componenti separate:

- Web Service: la parte del Framework responsabile di gestire le interazioni con il Front-End.
- Data Mining Plugin (DMPlugin): le funzionalità effettive, utilizzate durante gli esperimenti. Questi DMPlugin saranno implementati da sviluppatori, anche esterni al team di sviluppo.

- Interfaccia di amministrazione: utilizzata da un amministratore per installare e disinstallare funzionalità e per eseguire operazioni statistiche sulla suite.

Oltre a questi componenti principali, il Framework presenta altri package, ovvero:

- Xml Generator: un generatore di template, utilizzato per generare i file XML necessari per le comunicazioni con il Front-End.
- Data Types: gli oggetti che rappresentano entità fondamentali della suite, come un utente o un esperimento.
- Security: le classi utilizzate per la sicurezza della suite.
- Error Management: Un semplice contenitore degli errori che si possono verificare durante le operazioni della suite.

Tutto il Framework è stato implementato utilizzando il linguaggio Java. Le ragioni di questa scelta sono le seguenti:

- Semplicità: Java è un linguaggio semplice da utilizzare. Questo aiuta sia l'implementazione da parte del team, sia lo sviluppo di nuovi DMPlugin da parte di programmatori esterni.
- Portabilità: La Virtual Machine Java garantisce che il Framework possa essere installato su architetture diverse senza bisogno di effettuare modifiche al codice.
- Sicurezza: alcune caratteristiche di Java, come l'essere fortemente tipato e la gestione automatica della memoria, producono un codice più robusto, eliminando il rischio di operazioni dannose per il sistema, come l'uso improprio dei puntatori. [4]

Lo Use Case Diagram successivo rappresenta tutte le operazioni del Framework. Per dettagli maggiori sulle operazioni consultare le tabelle di Cockburn nell'appendice.

[4] Vedere anche http://java.sun.com

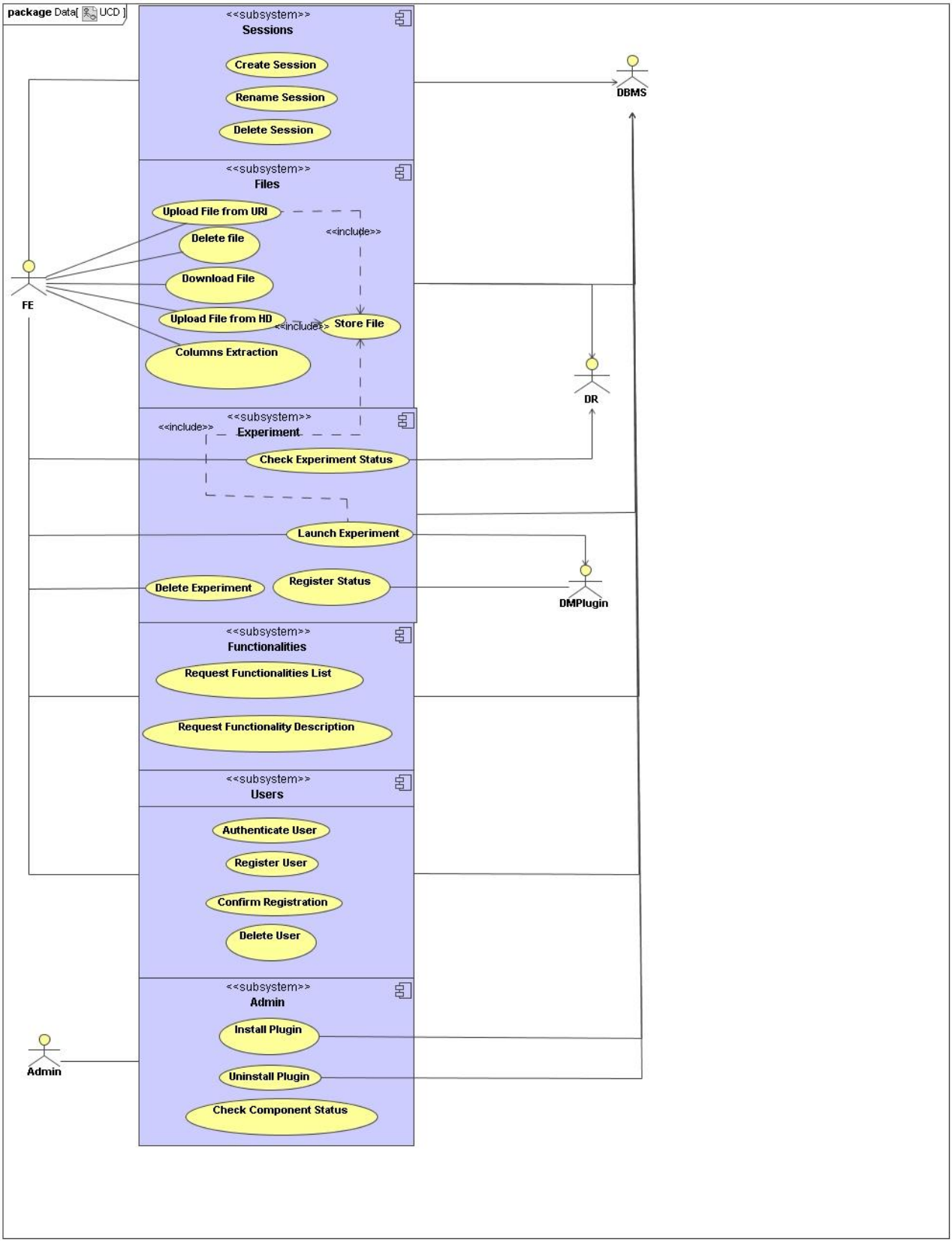


Figura 6: Framework Use Case Diagram

3.1. Web Service

3.1.1. Descrizione

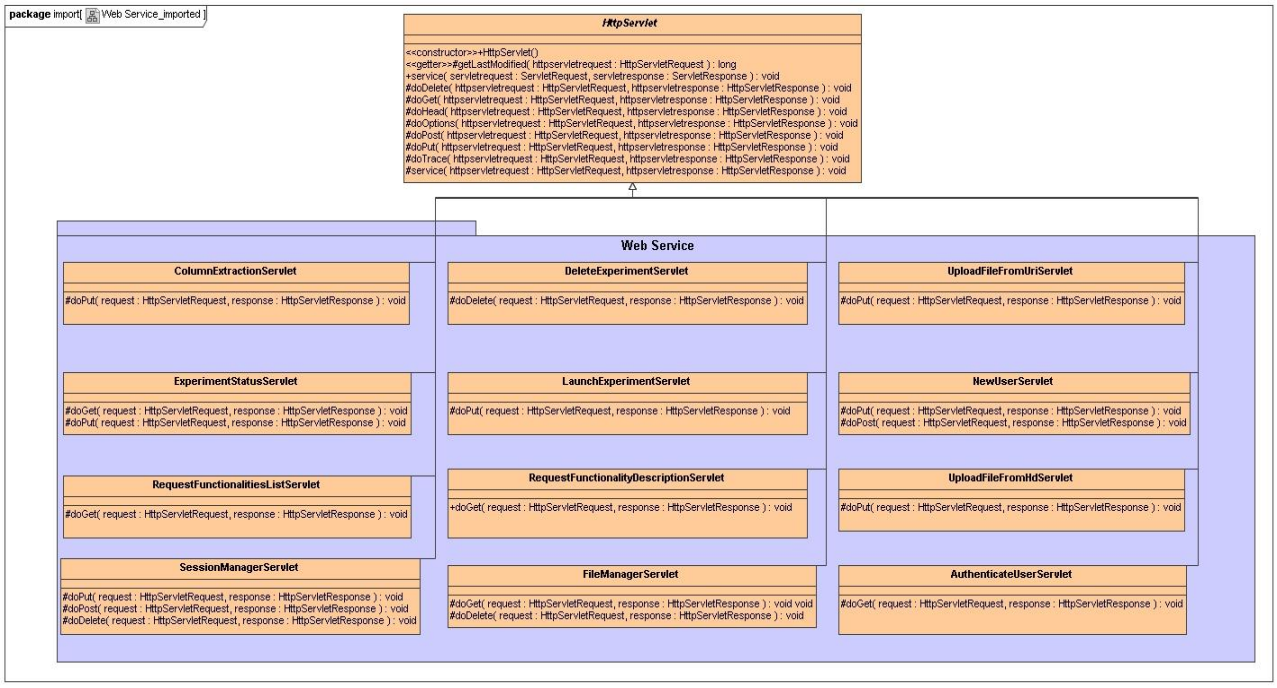


Figura 7: Web-Service Class Diagram

Il Web Service è la parte del Framework che si occupa di interagire con il Front-End. Esso è progettato secondo lo stile architetturale REpresentional State Transfer (REST). REST è uno stile architetturale client-server. I servizi web sono visti come risorse, ognuna delle quali è identificata da un Uniform Resource Identifier (URI). Elementi come i client, i server e le cache sono chiamati connettori. Un client può accedere a una risorsa per ottenere una sua rappresentazione; alcune risorse possono essere modificate. L'accesso dei client viene gestito tramite un'interfaccia standard, ad esempio i metodi http GET, PUT, POST e DELETE. L'architettura REST è cachable e stateless, due fattori che garantiscono buone prestazioni. Le rappresentazioni di una risorsa

possono essere inserite in una cache, riducendo il carico della rete. Il server non deve tenere traccia delle interazioni con i client, che sono atomiche, e può quindi, potenzialmente, gestire un numero maggiore di client.

Questa architettura si adatta perfettamente al Framework. Concetti chiave come sessioni, files ed esperimenti possono essere facilmente identificati come risorse, e ogni operazione su di esse risulta essere atomica. La semplicità di un'architettura REST consente una veloce implementazione sia del lato client che del lato server.

In Java, un' architettura Web Service RESTful può essere implementata in vari modi. Due possibilità sono l'utilizzo delle Servlet e del Framework Restlet.

Il Framework Restlet è un progetto open-source [5], che cerca di portare concetti REST, come risorse e connettori, in un ambiente a oggetti. In un primo tempo, il Framework si appoggiava alle API Servlet, ma si è successivamente evoluto, riuscendo a diventare una valida alternativa all'utilizzo delle Servlet per implementare un'architettura REST. Uno dei problemi principali del Framework Restlet è che, essendo un Framework relativamente nuovo e ancora in sviluppo, esiste poca documentazione, il che rende difficile il suo utilizzo.

La scelta finale è caduta sull'utilizzo delle Servlet, principalmente per via della vasta documentazione esistente, che rende la programmazione con le Servlet molto semplice. Il Framework Restlet include opzioni non ottenibili con le Servlet, come il supporto per le chiamate non bloccanti NIO. Queste opzioni però non sono risultate significative per il nostro Framework, il che ci ha portato a scartare definitivamente l'idea di utilizzare il Framework Restlet.

Per costruire l'architettura abbiamo, come prima cosa, individuato le risorse. A questo punto abbiamo definito una nuova Servlet, che specializza la classe HTTPServlet, per ogni risorsa, facendo l'override dei metodi corrispondenti alle operazioni su quella risorsa.

[5] Vedere http://www.restlet.org

La seguente tabella contiene la lista delle operazioni della suite.

Operazioni dell'Utente	Descrizione
Crea sessione	Permette ad un utente di creare una nuova sessione, nella quale potrà caricare files e creare nuovi esperimenti.
Rinomina Sessione	Permette ad un utente di rinominare una sessione.
Elimina Sessione	Permette ad un utente di eliminare una sessione
Carica file	Permette ad un utente di caricare un file sul File-Store. Il file può essere caricato dall'hard-disk dell'utente oppure da un URI.
Cancella File	Permette ad un utente di eliminare un file precedentemente caricato .
Scarica File	Permette ad un utente di scaricare un file .
Estrai Colonne	Consente ad un utente di estrarre delle determinate colonne ad un file di dati, e di inserirle in nuovo file.
Controlla stato di un esperimento	Permette all'utente di conoscere lo stato di un esperimento, compreso quali file di output sono già stati prodotti.
Crea Esperimento	Permette ad un utente di creare un nuovo esperimento.
Cancella Esperimento	Permette ad un utente di eliminare un esperimento, compresi i file da esso utilizzati.
Registrazione utente	Permette ad un utente di registrarsi.
Conferma registrazione	Dopo che la registrazione viene completata è necessario confermarla altrimenti, dopo un certo

	periodo di tempo, l'utente viene eliminato .
Autentica utente	Permette ad un utente di effettuare il log-in.

Operazioni del Plugin	Descrizione
Registra stato	Utilizzata da un plugin per registrare lo stato dell'esperimento ad esso associato, compreso quali file di output sono già stati prodotti.

Altre operazioni	Descrizione
Elimina utente	Operazione che parte automaticamente quando un utente si registra ma non conferma la propria registrazione entro un certo tempo.

Nella seguente tabella sono illustrate le risorse individuate, abbinate alle corrispondenti operazioni.

Nome Risorsa	URL	Operazioni
Session Manager	/sessions/session_name	• PUT: crea nuova sessione • POST: rinomina sessione • DELETE: elimina sessione
Request Functionalities List	/functionalities/	• GET: ottiene la lista delle funzionalità
Request Functionality Description	/functionalities/funcID	• GET: ottiene il documento di descrizione della funzionalità

File Manager	/files/ (URI="/some/URI")	• GET: scarica il file • DELETE: elimina il file
Upload File From HD	/files/ (URI="/some/URI")	• PUT: fa l'upload di un file da un hard disk
Upload File From URI	/files/ (URI="/some/URI")	• PUT: Fa l'upload di un file da un URI.
Column Extraction	/extract/file	• Put: estrae da un file determinate colonne, inserendole in un nuovo file.
Delete Experiment	/experiments/expID	• DELETE: elimina l'esperimento
Experiment Status	/esperiments/status/expID	• GET: Ritorna lo stato di un esperimento • PUT: modifica lo stato di un esperimento
Launch Experiment	/experiments/newExperiment/	• PUT: crea un nuovo esperimento.
New User	/users/newUser/userName (FILE=registration.xml for PUT) or (SSID="SomeID" for POST)	• PUT: registra un nuovo utente • POST: conferma la registrazione di un utente
Authenticate User	/users/authentication/ username (PASSHASH=somehash)	• GET: Autentica l'utente

3.1.2. Gli XML

Per gestire le comunicazioni con il Front-End, abbiamo scelto di utilizzare il formato XML. XML è un metalinguaggio generico, utilizzato per definire nuovi linguaggi di mark-up [6]. I suoi utilizzi sono molteplici e vanno dalla condivisione di dati strutturati alla loro serializzazione. Un documento XML contiene una serie di elementi, delimitati da etichette e strutturati secondo un albero. A ogni etichetta possono essere associati degli attributi per aumentare il suo livello di dettaglio. Un documento XML può essere ben formato e valido: è ben formato quando segue le regole sintattiche di XML. Alcune delle regole più importanti sono:

- Ogni elemento non vuoto deve essere delimitato da un'etichetta di apertura e da una di chiusura.
- Gli elementi possono essere composti ricorsivamente, ma ogni elemento deve essere completamente contenuto nei suoi genitori.

Ad esempio il documento seguente non sarebbe considerato ben formato:

<albero>

<frutto>

</albero>

</frutto>

La sua forma corretta sarebbe la seguente:

<albero>

<frutto>

< /frutto >

</albero >

[6] Vedere http://www.w3.org/XML/

Un secondo livello di correttezza di un documento XML è la sua validità. Un documento XML è definito valido se segue ulteriori regole semantiche, che permettono ad uno sviluppatore di definire il linguaggio secondo le sue necessità. Queste regole semantiche sono contenute in uno schema. Due esempi di formato di schema sono Document Type Definition (DTD) e XML Schema.

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.

XML Schema è stato creato come successore di DTD. 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.

XML possiede numerosi vantaggi che ci hanno portato a sceglierlo:

- Essendo un semplice documento di testo è altamente portatile.
- E' molto flessibile e permette di definire con semplicità i propri dati.
- Dato che un documento segue delle regole sintattiche ben precise, si può automatizzare la sua procedura di validazione.
- E' basato su standard internazionali.

Alcuni svantaggi sono:

- La sua struttura gerarchica pone seri problemi per strutture dati più complesse.
- La sua sintassi è molto ridondante e può porre limiti di efficienza sia per il volume dei dati, sia per il tempo necessario per elaborarli.

Nella nostra applicazione i vantaggi superano gli svantaggi. Abbiamo inoltre scelto di adottare XML Schema per la definizione dei vincoli dei documenti, preferendo la sua flessibilità maggiore rispetto a DTD. Lo schema da noi adottato è quello VO-Table[7].

3.1.3. Lo Schema VO-Table

VO-Table è uno schema, strutturato in modo da rappresentare in maniera efficiente dati tabulari. Alcune delle caratteristiche più importanti del VO-Table sono:

- Separazione di dati e meta-dati. I meta-dati, ovvero i nomi delle colonne della tabella, vengono definiti in una sezione precedente ai loro valori effettivi. VO-Table pone una grossa enfasi sui meta-dati, 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. Se è necessario conoscere solo la struttura della tabella, non è necessario scorrere tutto il documento, ma basta fermarsi alla fine della sezione dei meta-dati. Nel caso in cui sia necessario costruire una propria rappresentazione della tabella del documento, è possibile creare la struttura mentre si scorrono i meta-dati per poi assegnare, in maniera semplice i valori quando si giunge alla sezione dati.
- Struttura gerarchica. Gli elementi di una VO-Table possono essere combinati in maniera ricorsiva, in modo da rappresentare concetti più complessi.
- L'utilizzo degli Unified Content Descriptors (UCD). A ogni colonna di una tabella viene associato un UCD, che rappresenta il suo tipo di dato. Un client può fare quindi il parsing di un XML senza sapere il nome o la posizione effettiva delle colonne, ma limitandosi a considerare gli UCD a cui è interessato.

[7] Vedere http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/IvoaVOTable

All'inizio di ogni schema VO-Table, dopo un header contenente informazioni sulla versione di XML utilizzata, viene posto l'elemento VOTABLE. Tra i figli 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.

Dato che il Framework deve trattare prevalentemente dati astronomici, VO-Table rappresenta la soluzione ideale, essendo strutturato proprio per questo scopo. Abbiamo preferito utilizzare un singolo schema, in modo da semplificare sia la generazione, sia la validazione dei documenti XML, quindi ogni documento utilizzato nel Framework aderisce a VO-Table.

Il Framework dovrà lavorare con 6 diversi file XML, ovvero:

- Functionalities List: rappresenta le funzionalità supportate dal Framework, con una descrizione di ognuna di esse. Può essere richiesta dal Front-End secondo le sue necessità, ad esempio ad intervalli di tempo. (vedere risorsa Request Functionalities List).
- Functionality Description: Rappresenta il documento di descrizione dettagliato di una funzionalità, contenente i suoi dati di input, output e informazioni sulle sue operazioni. Viene generato dal DMPlugin corrispondente e viene richiesto dal Front-End quando

ottiene una lista di funzionalità contenente funzionalità nuove o aggiornate (vedere risorsa Request Functionality Description).

- Session Manager: Richiesto dal Front-End dopo il log-in di un utente. Contiene informazioni sulle sessioni create da un utente, sui suoi esperimenti e sui file da lui caricati sul file-store (vedere risorsa Authenticate User).
- Interactive Report: Rappresenta informazioni riguardanti un esperimento e il suo stato, compresi i suoi file di output. Viene richiesto dal Front-End dopo il log-in di un utente o a intervalli di tempo (vedere risorsa Get Experiment Status).
- Meta-Data: contiene le informazioni riguardante i meta-dati di un file. Viene creato dal Framework dopo che il Front-End fa l'upload di un file e viene contenuto interamente sul Database.
- Status Report: contiene informazioni riguardanti lo stato di un esperimento e dei suoi file di output. Viene preparato dal DMPlugin ed inviato al Framework.

Consultare l'appendice per esaminare gli XML corrispondenti.

3.1.4. Il generatore di Template

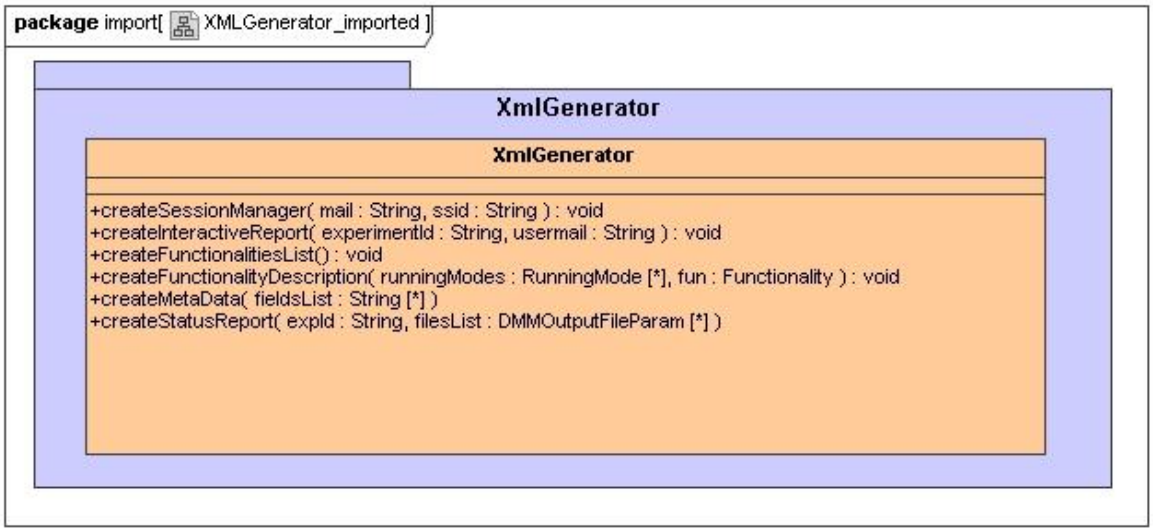


Figura 8: Xml Generator Class Diagram

Per semplificare la creazione dei files XML, abbiamo deciso di associare un template a ogni documento. Un template è semplicemente un modello generico rappresentante la struttura di un qualunque documento. Definendo i template secondo i vincoli dello schema VO-Table è possibile creare a partire da questi, tramite un apposito generatore, documenti XML ben formati e validi. Il generatore da noi adottato è Freemarker[8]. Freemarker è un'applicazione generica che consente di generare output testuali di qualunque genere a partire da un template, definito con un semplice linguaggio di programmazione, che consente però di creare modelli estremamente flessibili. 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.

Un template Freemarker è costituito da quattro diversi elementi:

[8] http://freemarker.sourceforge.net/

- Il normale testo del documento.
- Le interpolazioni. Ogni volta che Freemarker incontra una sequenza del tipo ${...} sostituisce la variabile contenuta tra le parentesi graffe con il suo valore.
- Le etichette FLT. Gli elementi con queste etichette rappresentano le istruzioni di Freemarker, chiamate direttive, e non vengono visualizzati come output. Un utente può anche definire le proprie direttive, in modo da costruire operazioni più complesse utilizzabili in seguito nel template.
- I commenti. Tutto il testo compreso tra <#-- e --> verrà ignorato da Freemarker.

Freemarker presenta una notevole lista di direttive. Le uniche che abbiamo avuto la necessità di utilizzare sono però #if, che permette di inserire condizioni nel template, e #list, che permette di scorrere una lista di variabili.

3.1.5. Il Parsing di un documento XML

Quando il Framework, o il DMPlugin ricevono un documento XML generato dal Front-End, questi devono effettuare il parsing del documento per ricavarne delle informazioni. Esistono diversi metodi per effettuare questo parsing: i due più utilizzati sono Document Object Model (DOM) e Simple Api for XML (SAX).

Un parser DOM effettua le sue operazioni caricando l'intero documento in memoria sotto forma di un albero. A questo punto è possibile visitare i nodi di quest'albero utilizzando le API DOM. Il vantaggio principale del DOM è sicuramente la sua flessibilità. Un programmatore può accedere a qualunque sezione del documento, secondo le sue preferenze. I suoi svantaggi principali sono però due. Per documenti grandi il caricamento in memoria può essere piuttosto pesante. Inoltre è necessario che il programmatore conosca bene la struttura del documento, altrimenti gli può risultare complicato scorrerlo.

Un parser SAX ha un approccio opposto. Durante il parsing del documento vengono lanciati eventi quando si verifica una delle seguenti condizioni:

- L'inizio o la fine del documento.
- L'inizio o la fine di un elemento.
- Vengono incontrati dei caratteri al di fuori di un etichetta.

Un programmatore deve definire un proprio gestore, che preveda dei metodi per trattare gli eventi per lui significativi. Il grande vantaggio di SAX è la sua semplicità. Non è necessario conoscere la struttura del documento per effettuare il parsing, basta sapere come trattare gli eventi interessanti. SAX può però essere inefficiente. Il documento può essere consultato solo dal principio verso la fine. Non esiste la possibilità di consultare elementi precedenti a quello che si sta scandendo in un determinato istante. Potrebbe essere necessario scandire più volte uno stesso documento in caso di parsing più complessi.

Sia per il DMPlugin che per il Framework abbiamo preferito la semplicità di SAX alla potenza del DOM. Infatti, nelle nostre operazioni non abbiamo ritenuto significativi i vantaggi che offre quest'ultimo.

3.1.6. La Sicurezza della Suite

In questa prima release, il team di sviluppo non ha ritenuto necessario introdurre meccanismi di sicurezza eccessivamente complessi. Si è preferito, infatti, concentrarsi sugli aspetti più complessi della suite, relegando a versioni successive l'approfondimento della sicurezza. Sono comunque stati introdotti tre diversi meccanismi per proteggere la suite:

- Verifica degli ip: ogni volta che il Framework viene contattato, verifica l'indirizzo ip del oggetto chiamante per accertarsi che sia uno dei Front-End autorizzati. In questo modo si limita la possibilità che un qualunque utente possa accedere direttamente al Framework, eventualmente effettuando operazioni dannose.

- Security Session Id (SSID): dopo che un utente effettua un log-in, il Framework genera un SSID: una stringa di 8 caratteri, criptati in md-5. A ogni interazione successiva di un utente il Front-End dovrà inviare il suo SSID, che verrà confrontato con quello generato dal Framework per capire se l'utente è autorizzato.
- Security Experiment Id (SEID): dopo che viene creato un nuovo esperimento, viene generato un SEID che segue le stesse modalità del SSID. Questo evita che applicazioni malevole si possano spacciare per DMPlugin e accedere al sistema.

Dato che la suite è pensata principalmente per scopi di ricerca scientifica, non si pensa che siano necessari meccanismi eccessivamente complessi. Nonostante ciò, in futuro, sarà possibile che questo aspetto venga analizzato con più attenzione.

3.2. I Data Mining Plugin ed i Data Mining Models

3.2.1. Descrizione

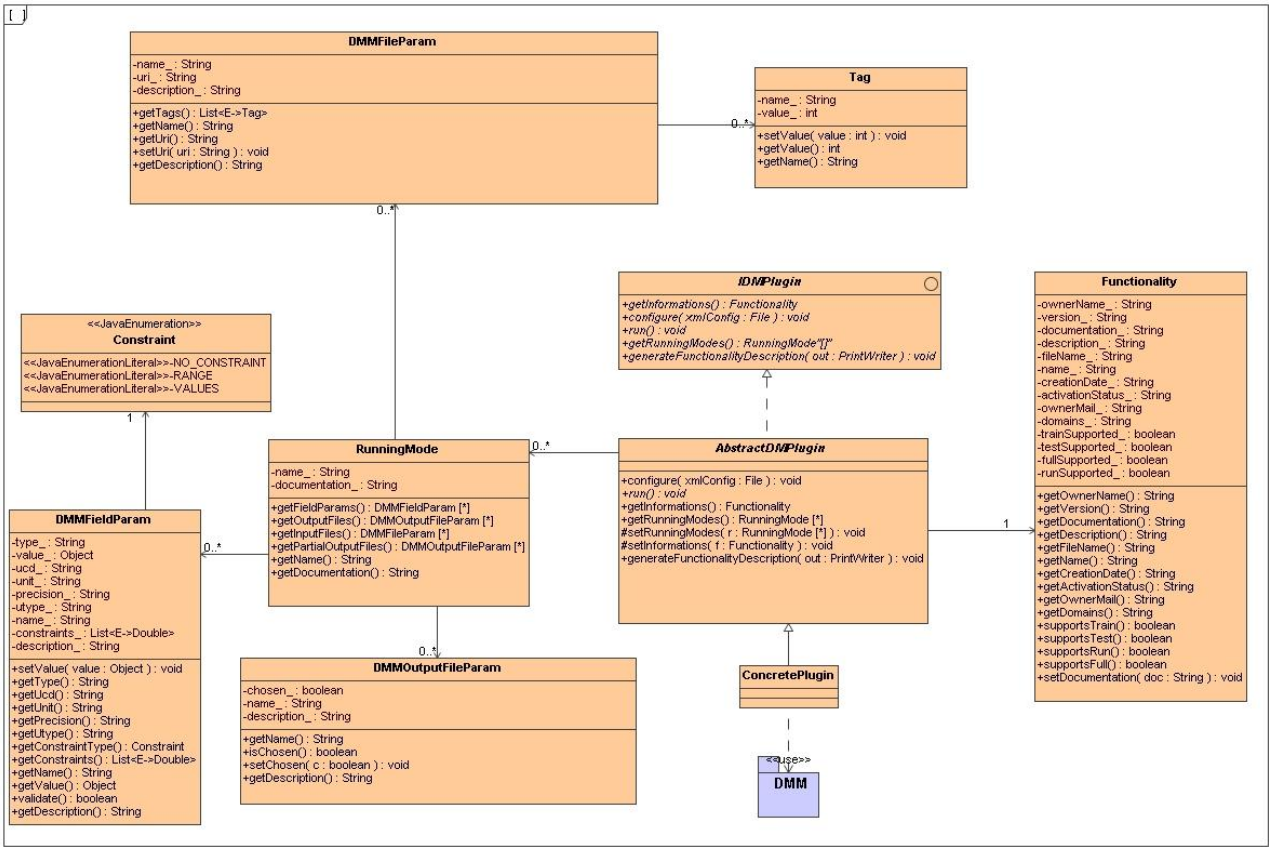


Figura 9: DMPlugin

I Data Mining Plugin (DMPlugin) sono le funzionalità offerte dalla suite per effettuare gli esperimenti di Data Mining. Strutturare le funzionalità sotto forma di plugin permette di estendere in maniera semplice la suite, senza la necessità di ricompilare il Framework. I DMPlugin possono essere realizzati da sviluppatori esterni. Ogni DMPlugin dovrà implementare un'interfaccia, che permetterà ai componenti esterni di configurarlo, di eseguire l'esperimento associato al plugin e di ottenere informazioni da esso. Durante la fase di analisi abbiamo individuato alcune componenti che accomunano ogni DMPlugin, e abbiamo quindi provveduto a costruire un classe astratta che le implementi. Ogni DMPlugin avrà una serie di informazioni, contenute in un oggetto di tipo

Functionality, e una lista di modalità di esecuzione, ad esempio train e test. Ogni modalità di esecuzione dovrà a sua volta contenere, oltre ad informazioni generale su di essa, quattro diverse liste:

- Una lista di parametri di input. Ogni parametro potrà avere uno fra tre diversi vincoli ovvero: “Nessun vincolo”, “vincolo di campo”, “vincolo di valori”. La procedura *validate* controllerà che il vincolo del parametro sia rispettato.
- Una lista di file di input. Generalmente questi file sono insiemi di dati. Le colonne di questi insiemi di dati devono essere etichettate dall’utente secondo specifiche richieste dello sviluppatore del DMPlugin. Queste specifiche saranno inserite nella classe Tag.
- Una lista di file di output e una di file di output parziale. L’utente potrebbe non essere interessato a tutti gli output, quindi questa classe conterrà un parametro booleano in cui inserire la scelta dell’utente.

Per effettuare un esperimento un DMPlugin deve utilizzare dei Data Mining Models (DMM). L’utilizzo di un DMM deve essere estremamente semplice per lo sviluppatore. Questi dovrà limitarsi a chiamare i metodi offerti dal DMM, senza sapere con precisione il funzionamento del modello. Per fare ciò è stato necessario effettuare un lavoro di ingegnerizzazione sui DMM, in modo da costruire una gerarchia di classi. Questa gerarchia parte da concetti astratti e li specializza sempre di più fino ad arrivare agli algoritmi effettivi. Durante il processo di ingegnerizzazione abbiamo incontrato un problema: in una prima analisi il modo più semplice di rappresentare alcune delle relazioni sembrava essere l’ereditarietà multipla. Questo meccanismo non è però supportato in java ed anche in altri linguaggi può produrre grossi problemi. Ad esempio se una classe eredita due metodi con la stessa firma da due classi diverse questo produce un’ ambiguità difficile da risolvere.

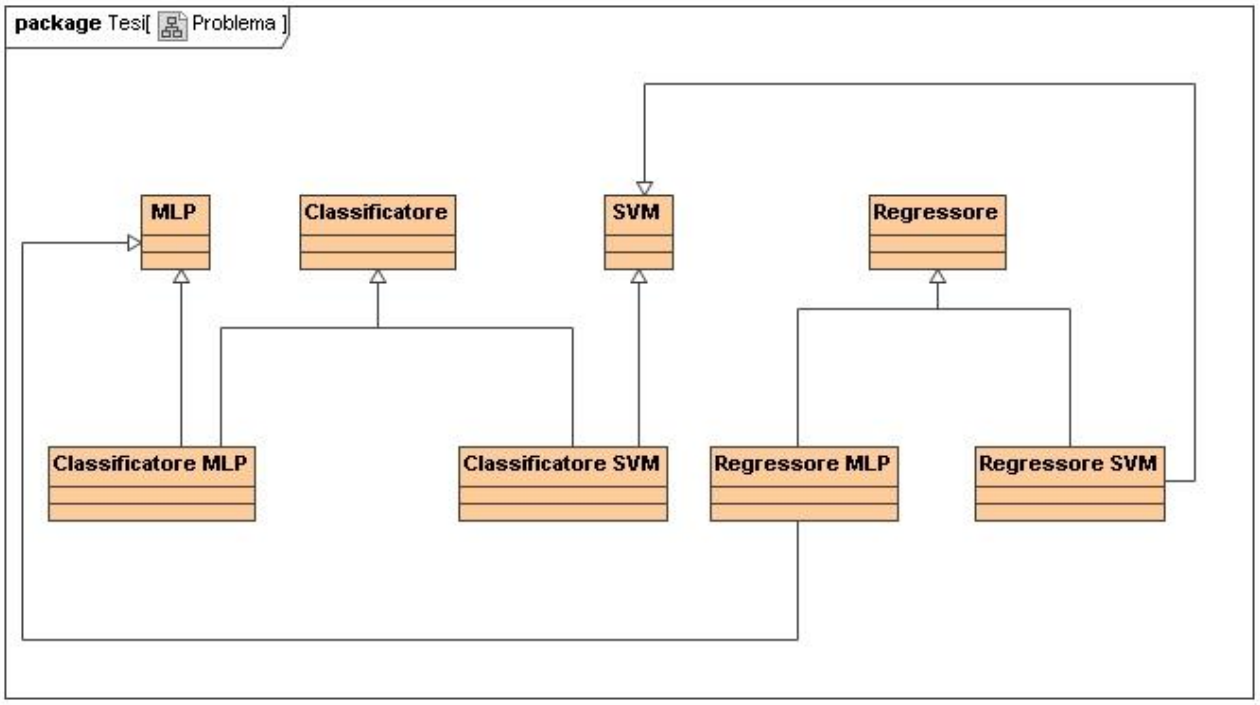


Figura 10: Problema iniziale DMM

Una prima soluzione analizzata è stata quella di eliminare l'ereditarietà multipla nel seguente modo.

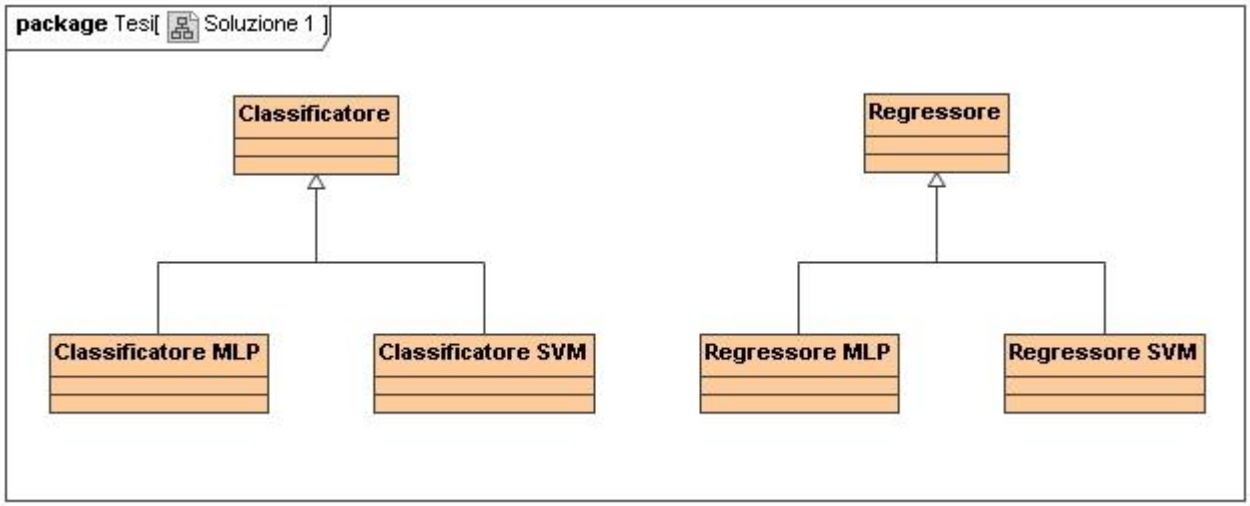


Figura 11: Prima soluzione DMM

Questa soluzione implica però un enorme duplicazione di codice, che non è sicuramente positiva, soprattutto dal punto di vista della manutenzione, ed è stata presto scartata.

Una successiva soluzione analizzata consisteva nello sfruttare le interfacce per simulare l'ereditarietà multipla.

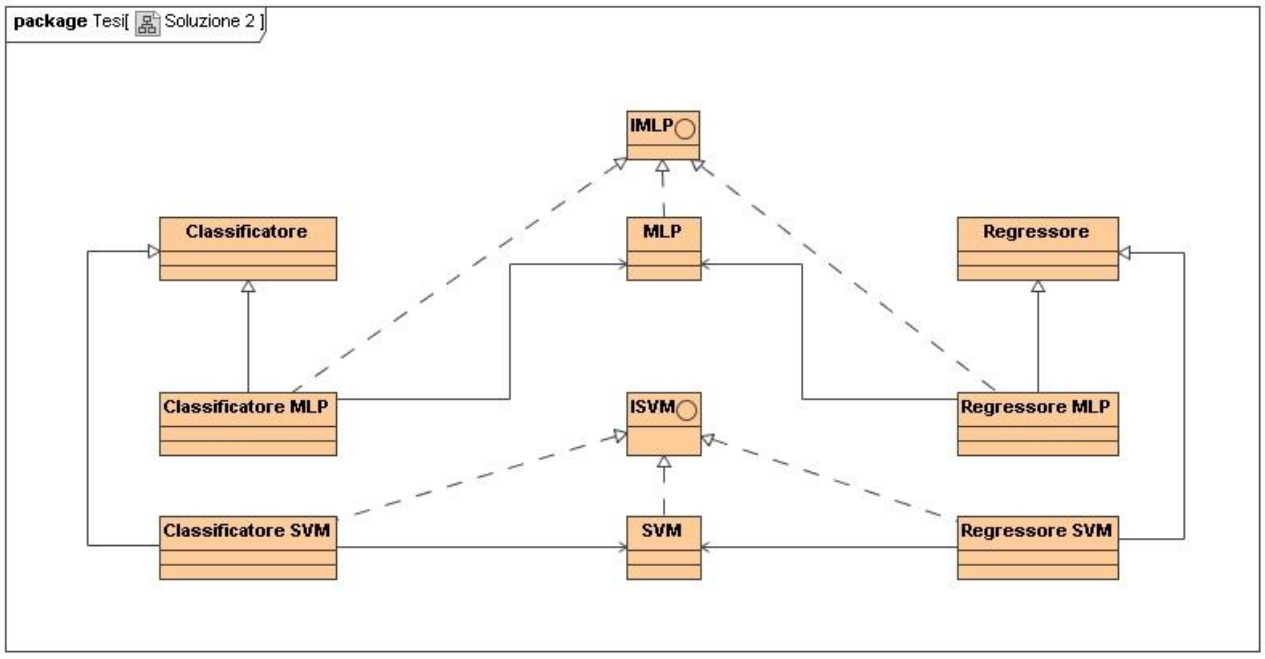


Figura 12: Soluzione 2 DMM

In questa soluzione un classificatore MLP avrebbe ereditato direttamente dalla classe Classificatore, implementando però l'interfaccia IMLP. Per implementare i metodi dell'interfaccia il Classificatore MLP avrebbe utilizzato un'istanza di MLP. Esempio:

```
class MLP implements IMLP {
public void train() {
....
}
}
```

```
class ClassificatoreMLP extends classificatore implements IMLP {
        MLP mlp;
        public void train() {
                mlp.train();
        }
```

Questa soluzione risolve il problema dell'ereditarietà multipla, ma non è sicuramente l'approccio ideale da un punto di vista ingegneristico. Ogni volta che viene aggiunto un nuovo modello è necessario inserire una classe per il modello, una nuova interfaccia associata al modello e tutto un insieme di classi associate.

La soluzione finale scelta adotta il design pattern bridge ed è rappresentata nella figura seguente.

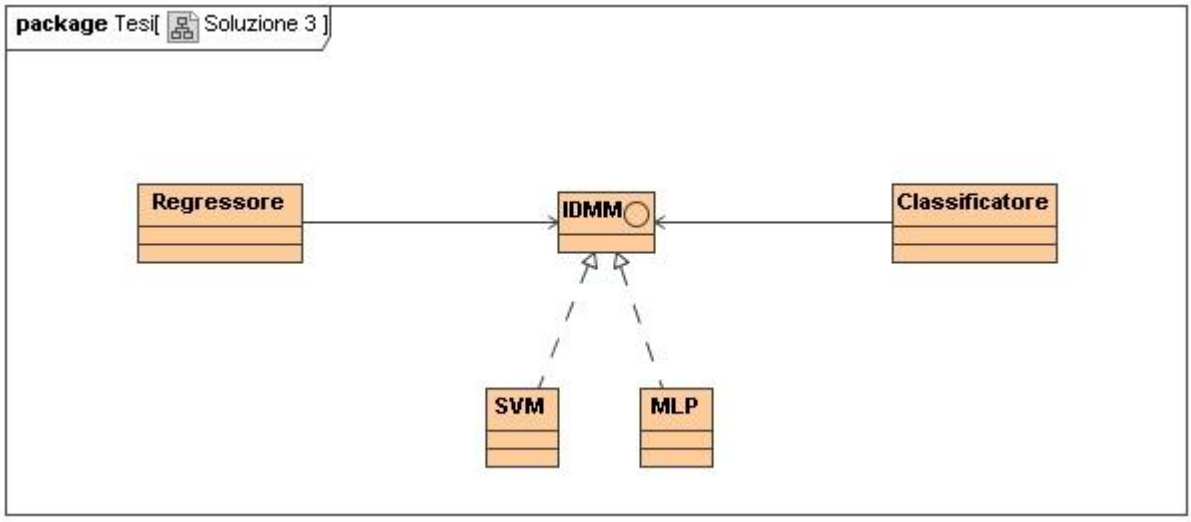


Figura 13: Soluzione 3 DMM

Questa soluzione è la più efficiente da un punto di vista ingegneristico: se viene aggiunto un nuovo modello è necessario introdurre una sola classe nuova. Non esistendo un modo perfetto per risolvere questo problema, anche questa soluzione presenta dei difetti: i modelli devono accordarsi su un'interfaccia comune ma i loro metodi potrebbero necessitare di parametri diversi. I parametri dei modelli sono quindi astratti da una nuova classe DMMParam. I metodi pubblici dei modelli dovranno accettare un array di oggetti DMMParam. Questo significa che a tempo di compilazione non è possibile rilevare errori nell'utilizzo dei metodi dei modelli, il che potrebbe inserire problemi difficili da individuare successivamente. Per minimizzare questo problema un DMPlugin utilizzerà solo una volta i DMMParam per settare il modello. A questo punto il modello controllerà i parametri per verificare che siano del giusto numero e tipo, generando un eccezione in caso di errori. Il DMPlugin non dovrà passare ulteriori parametri per utilizzare gli altri metodi dei modelli, che non necessiteranno di parametri nella loro firma.

3.2.2. Il ciclo di vita di un Data Mining Plugin

Il ciclo di vita di un DMPlugin inizia quando il Framework riceve una richiesta dal Front-End riguardante la creazione di un nuovo esperimento. A questo punto, il Framework fa il parsing del file XML ricevuto, in modo da avere sufficienti informazioni per instanziare e configurare il DMPlugin corretto. Durante la configurazione il DMPlugin ottiene i file di input e crea i DMM necessari per l'esecuzione successiva. Il Framework utilizza quindi il Driver Management System per iniziare l'esecuzione dell'esperimento. Se viene utilizzato il driver Stand Alone, allora questo si limita ad eseguire il metodo run del DMPlugin e l'esperimento può iniziare. Se invece viene utilizzato il driver per GRID il processo è più complesso. Il DMPlugin deve infatti venire trasferito su un Worker Node, insieme a tutti gli oggetti che utilizza. Per effettuare questa migrazione abbiamo analizzato diverse soluzioni. La prima soluzione analizzata è la serializzazione. La serializzazione è un processo che trasforma un oggetto in un flusso di byte, che possono essere salvati in un file. In

questo modo è possibile creare una copia identica dello stato di un oggetto, compresi i valori dei suoi attributi. L'operazione opposta è chiamata deserializzazione, e ritrasforma il flusso di byte nell'oggetto originale. Utilizzando la serializzazione l'oggetto può essere trasferito sul Worker Node, dove sarà lanciato un eseguibile che lo deserializzerà e lancerà l'esperimento. Il vantaggio di questo metodo è la sua semplicità. Il processo però può avere degli svantaggi. Quando si prova a serializzare oggetti complessi, la serializzazione può diventare molto complessa e provocare errori. Se il processo non viene controllato, in alcuni casi potrebbe causare una reazione a catena, che porterebbe ad un'enorme mole di oggetti serializzati, che rendono l'operazione poco efficiente. Un altro svantaggio della serializzazione è la perdita dell'incapsulamento dei dati e quindi l'esposizione delle sezioni private di un oggetto. Questo può essere svantaggioso per oggetti commerciali e, in alcuni casi, può essere necessario criptare i dati serializzati.

Un'alternativa alla serializzazione è di costruire i DMPlugin come eseguibili, dotati quindi di un main, e di inviare direttamente le classi corrispondenti sul Worker Node. Questo approccio presenta però uno svantaggio, ovvero l'impossibilità di interagire con il DMPlugin prima della migrazione. Questo comporterebbe la trasformazione dei dati di configurazione in parametri da passare al main, che rendono più complicato l'utilizzo di un DMPlugin ed eliminano il controllo della configurazione a tempo di compilazione.

Dopo aver analizzato vantaggi e svantaggi dei due metodi abbiamo deciso di adoperare la serializzazione, cercando di prevenire e limitare i suoi svantaggi. Dopo che il Driver ha serializzato il DMPlugin in un file, questo file può essere inviato sul Worker Node, dove verrà fatto partire un eseguibile che lo deserializzerà, riportando il DMPlugin nelle sue condizioni precedenti. A questo punto l'eseguibile farà partire l'esperimento invocando il metodo run del DMPlugin. Il Framework non può contattare con semplicità il DMPlugin una volta che l'esperimento è stato lanciato perché non può sapere dove si trova effettivamente. Sarà il DMPlugin stesso ad avere il compito di

registrare i propri file di output, quando sono pronti, e di avvisare successivamente il Framework, indicando lo stato di avanzamento della sua esecuzione.

Il Framework può ottenere alcune informazioni sullo stato del DMPlugin anche utilizzando il driver, che potrà dire se il processo è in esecuzione, se è in coda o se si sono verificati dei problemi.

I seguenti diagrammi mostrano alcuni dettagli riguardanti il ciclo di vita di un DMPlugin.

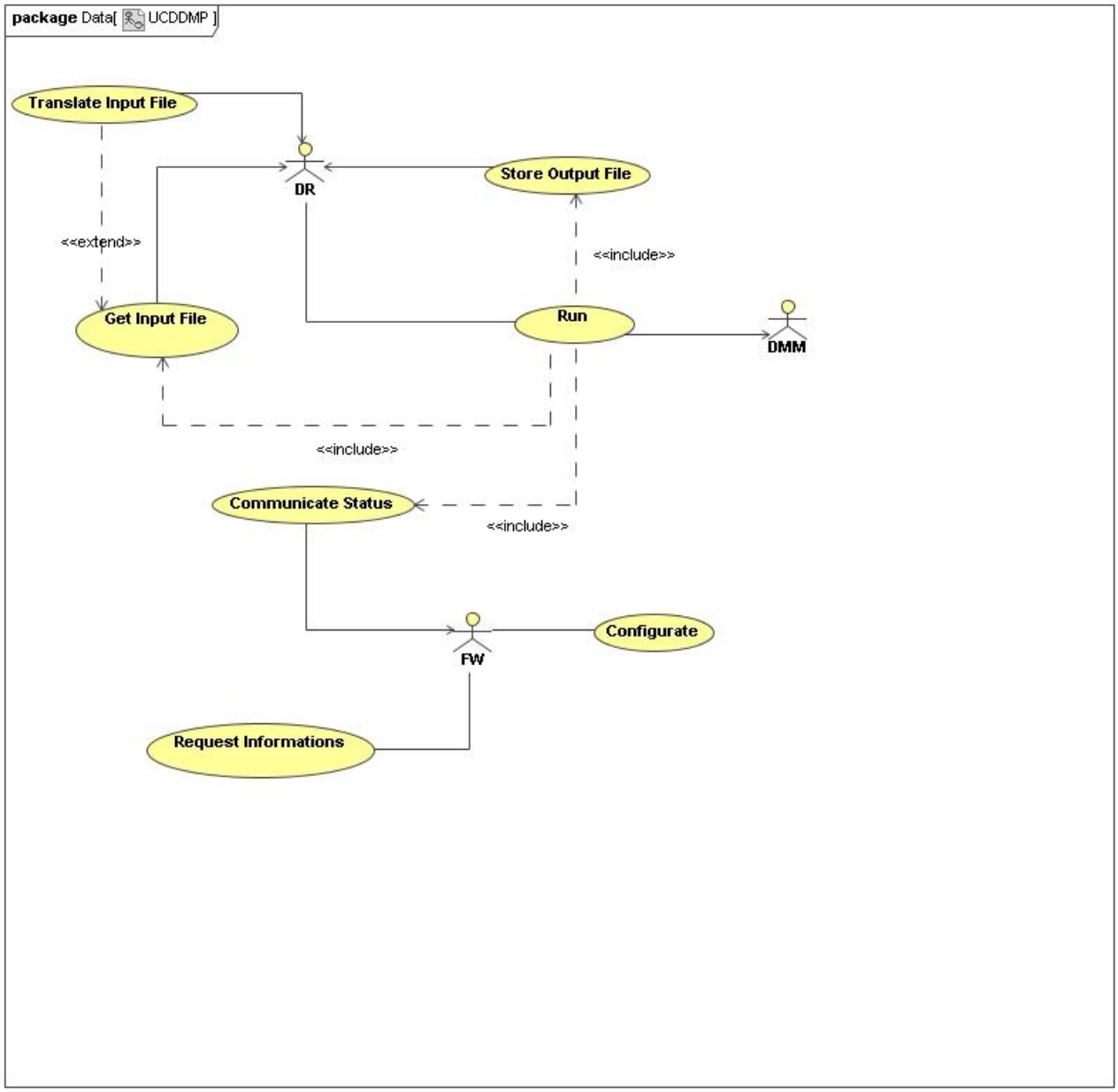


Figura 14: DMPlugin Use Case Diagram

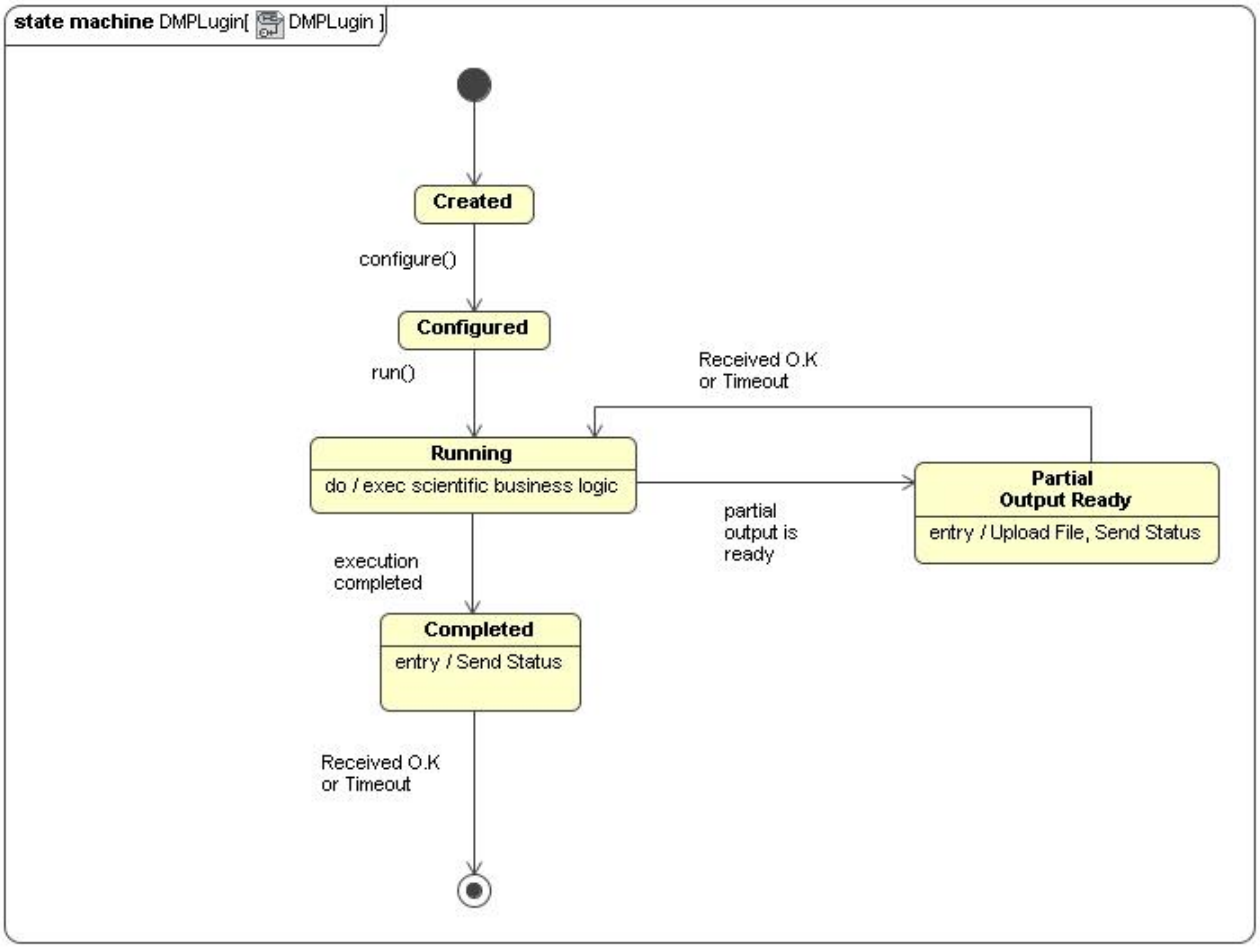


Figura 15: StateChart DMPLugin

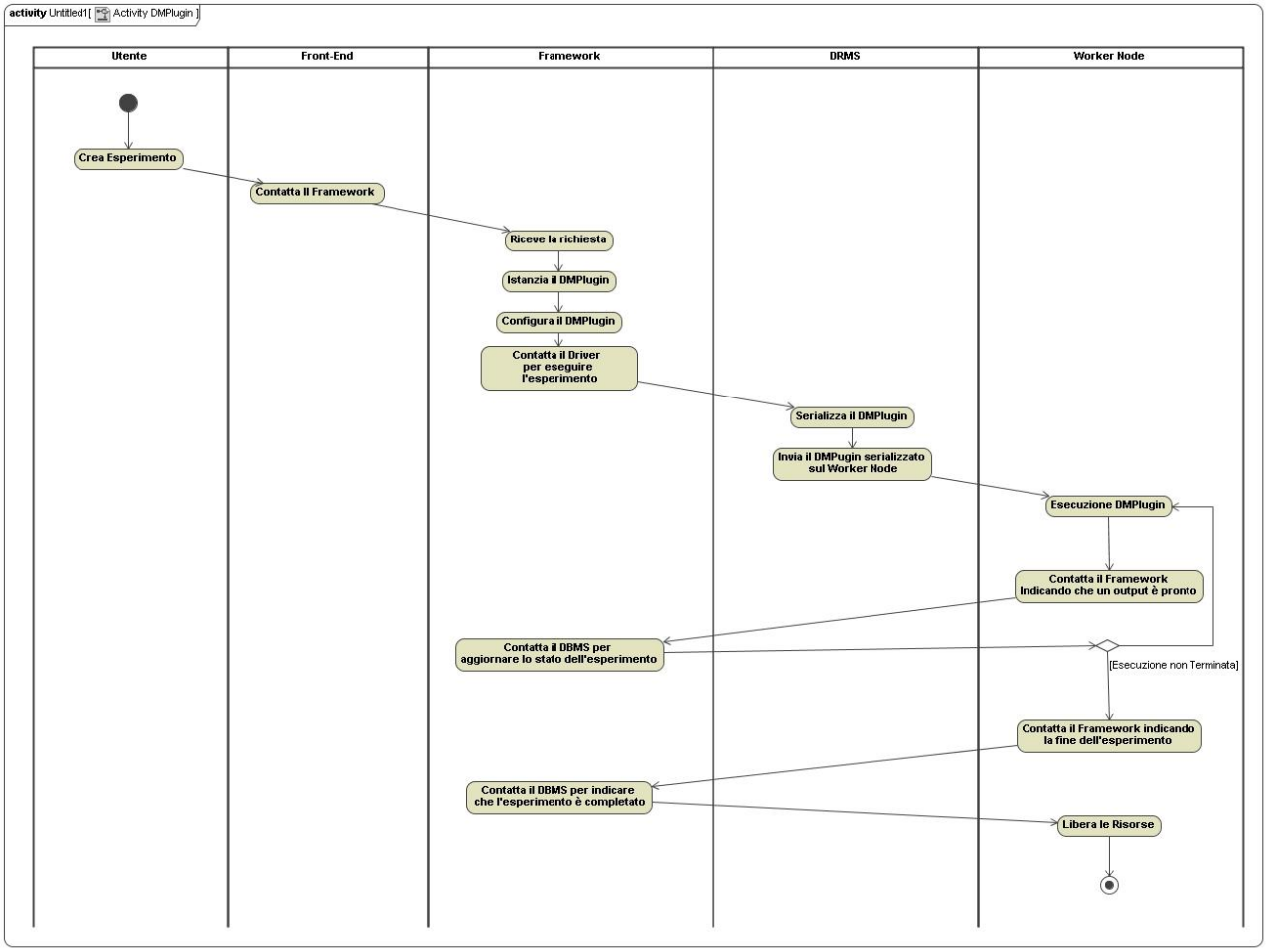


Figura 16: Activity Diagram DMPlugin GRID

3.3. La sezione di Amministrazione

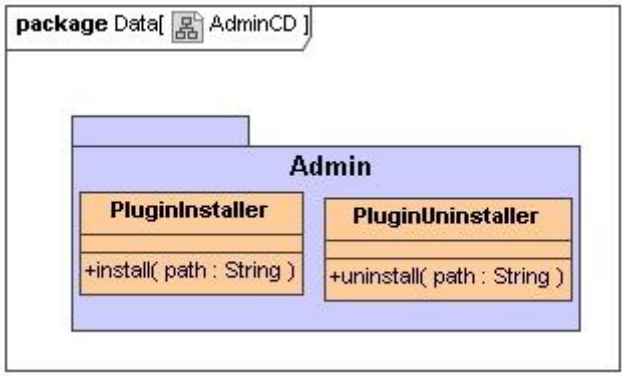


Figura 17: Admin Class Diagram

La sezione di amministrazione presenta una semplice interfaccia, basata su riga di commando, che permette a un Admin di installare o disinstallare nuove funzionalità. Al momento questa interfaccia è estremamente rudimentale. In futuro però si prevede la possibilità di inserire un'interfaccia grafica, in modo da semplificare il lavoro dell'Admin. Si pensa inoltre che sia necessario fornire meccanismi che consentano di monitorare l'uso della suite. Alcune delle operazioni che potrebbero essere implementate nel futuro sono le seguenti:

- Analisi del numero di utenti che utilizza la suite e della loro frequenza di accesso al sistema.
- Analisi del numero di funzionalità presenti nella suite e della percentuale di utilizzo di ognuna di esse.
- Analisi della dimensione del File Store.

Utilizzando queste operazioni è possibile sia semplificare le operazioni di diagnostica , sia analizzare quali funzionalità della suite incrementare, ad esempio analizzando quali funzionalità vengono utilizzate più comunemente.

4. Conclusioni

La prima release della suite di Data Mining mira a offrire una semplice base, da espandere però in futuro. Tra i punti analizzati per gli sviluppi futuri ci sono:

- Espansione delle Front-End. Si pensa di inserire nuove funzionalità che rendano più semplici e comode le operazioni di un utente, come inserire in un log le sue operazioni e memorizzare informazioni sugli esperimenti, che consentano facilmente di creare esperimenti simili ad altri già conclusi.
- Sviluppo del DRMS. Al momento ogni Framework possiede un solo Driver, associato a GRID o ad una macchina Stand-Alone. In futuro è prevista la possibilità di inserire più driver su uno stesso Framework, che comunicherà quindi con un DRMS, che sceglierà il driver più appropriato per la determinata situazione.
- Implementazione di nuove politiche di sicurezza. Il Framework al momento utilizza un meccanismo di sicurezza estremamente semplice. Nonostante la suite non necessiti di meccanismi di sicurezza avanzati, sarà comunque necessario approfondire questo aspetto.
- Introduzione di nuove funzionalità. La struttura a plugin del Framework consente di espandere facilmente le sue funzionalità, che potranno essere realizzate da sviluppatori esterni ed introdotte ad intervalli di tempo nella suite.
- Aggiornamento della sezione di amministrazione. Sarà necessario aggiungere nuove operazioni a questa sezione, che permettano ad un admin di avere più informazioni sul sistema. Probabilmente sarà aggiunta un'interfaccia grafica che permetta un lavoro più semplice da parte dell'admin.

5. Appendice

5.1. Diagrammi di Cockburn

In questa sezione ci sono i diagrammi di Cockburn delle operazioni del Framework.

5.1.1. Sessione

Use Case # 1	Crea Sessione
Goal in Contex	Il Front-End vuole creare una nuova sessione utente
Scope & Level	
Preconditions	
Success End Conditions	Il Front-End riceve il nuovo id di sessione
Failed End Conditions	Il Front-End riceve un K.O.
Primary Actor	Front-End
Trigger	Il Front-End invia una richiesta “crea nuova sessione”

Description	Step n°	Actor	System
	1	Invia una richiesta “crea nuova sessione”	
	2		Riceve la richiesta e contatta il DBMS, chiedendo il nuovo ID di sessione e comunicando il nome della
	3		Riceve il nuovo ID di sessione e

			lo invia al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS, invia un errore al Front-End

Use Case # 2			Rinomina Sessione
Goal in Contex			Il Front-End vuole rinominare una sessione esistente
Scope & Level			
Preconditions			
Success End Conditions			La session viene rinominata. Il Front-End riceve un O.K.
Failed End Conditions			La session non viene rinominata. Il Front-End riceve un K.O.
Primary Actor			Front-End
Trigger			Il Front-End invia una richiesta “rinomina session”
Description	Step n°	Actor	System

	1	Invia una richiesta “rinomina sessione”, che include l’id della sessione ed il nuovo nome	
	2		Riceve la richiesta e contatta il DBMS, chiedendogli di rinominare la sessione
	3		Riceve un O.K. dal DBMS
	4		Invia un O.K. al Front-End
Extensions 1: Sessione non trovata	Step n°	Actor	System
	3a		Riceve un errore di sessione non trovata dal DBMS. Invia un K.O. al Front-End.
Extensions 1: Errore DBMS	Step n°	Actor	System
	3b		Non riesce a contattare il DBMS, invia un errore al Front-End

Use Case #18	Elimina sessione
Goal in Contex	Il Front-End vuole eliminare una sessione
Scope & Level	
Preconditions	
Success End Conditions	La session viene eliminate
Failed End Conditions	La sessione non viene eliminata
Primary Actor	Front-End
Trigger	Il Front-End invia una richiesta “elimina sessione”

Description	Step n°	Actor	System
	1	Invia una richiesta “elimina sessione”, che include l’id della sessione	
	2		Riceve la richiesta e contatta il DBMS per rimuovere la sessione
	3		Riceve un O.K. dal DBMS. Invia un O.K. al Front-End.
Extension1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un K.O. al Front-

			End.
Extension 2: Sessione non trovata	Step n°	Actor	System
	3b		Riceve un errore “sessione non trovata”. Invia un K.O. al Front-End.

5.1.2. Funzionalità

Use Case # 3			Richiedi lista funzionalità
Goal in Contex			Il Front-End vuole ottenere la lista delle funzionalità
Scope & Level			
Preconditions			
Success End Conditions			Il Front-End riceve la lista delle funzionalità
Failed End Conditions			Il Front-End riceve un K.O.
Primary Actor			Front-End
Trigger			Il Front-End invia una richiesta “lista funzionalità”
Description	Step n°	Actor	System
	1	Invia una richiesta “lista di funzionalità”	
	2		Riceve la richiesta e contatta il DBMS, chiedendo la lista delle funzionalità

	3		Riceve la lista delle funzionalità
	4		Invia la lista al Front-End.
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un K.O. al Front-End

Use Case # 4			Richiedi descrizione funzionalità
Goal in Contex			Il Front-End vuole il documento di descrizione di una specifica funzionalità
Scope & Level			
Preconditions			
Success End Conditions			Il Front-End riceve il documento
Failed End Conditions			Il Front-End riceve un K.O.
Primary Actor			Front-End
Trigger			Il Front-End invia una richiesta “descrizione funzionalità”
Description	Step n°	Actor	System
	1	Invia una richiesta “descrizione funzionalità”, che include il nome della funzionalità richiesta.	
	2		Riceve la richiesta e contatta il DBMS, chiedendo il documento di descrizione.

	3		Riceve il documento.
	4		Invia il documento al Front-End.
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un K.O. al Front-End.
Extensions 2: Funzionalità non trovata	Step n°	Actor	System
	3b		Riceve un errore “funzionalità non trovata”. Invia un K.O. al Front-End.

5.1.3. Files

Use Case # 5			Download file
Goal in Contex			Il Front-End vuole scaricare un file
Scope & Level			
Preconditions			
Success End Conditions			Il Front-End riceve il file
Failed End Conditions			Il Front-End riceve un K.O.
Primary Actor			Front-End
Trigger			Il Front-End invia una richiesta “download file”
Description	Step n°	Actor	System
	1	Invia una	

		richiesta “download file”, che include l’URI del file.	
	2		Riceve l’URI e contatta il DBMS, controllando la flag di presenza
	3		Riceve la flag dal DBMS
	4		Contatta il DRMS, chiedendo il file
	5		Riceve il file e lo invia al Front-End.
Extensions 1: Errore DRMS	Step n°	Actor	System
	5a		Riceve un K.O. e lo invia all’utente.
Extensions 2: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un K.O. all’utente
Extensions 3: File non disponibile	Step n°	Actor	System
	4a		La flag ricevuta è “false”. Invia un K.O. all’utente

Extension 4: File non trovato	Step n°	Actor	System
	5b		Riceve un errore di “file non trovato”. Invia un K.O. all’utente

Use Case # 6	Upload file da HD
Goal in Contex	Il Front-End vuole fare l’upload di un file
Scope & Level	
Preconditions	
Success End Conditions	Il file viene caricato sul File Store ed il Front-End riceve un O.K.
Failed End Conditions	Il file non viene caricato sul File Store. Il Front-End riceve un K.O.
Primary Actor	Front-End
Trigger	Il Front-End invia una richiesta “Upload File da HD”

Description	Step n°	Actor	System
	1	Invia una richiesta di “upload file da HD”, che include il file	
	2		Fa lo store del file (riferirsi all’UC #7)

	3		Invia un O.K. al Front- End
Extensions 1: Errore DRMS	Step n°	Actor	System
	3a		UC #7 ritorna un errore DRMS. Invia un K.O. al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	3b		UC #7 ritorna un errore DBMS. Invia un K.O. al Front-End

Use Case # 7		Store File	
Goal in Contex		Un file deve essere salvato nel File Store	
Scope & Level			
Preconditions			
Success End Conditions		Il file viene salvato	
Failed End Conditions		Il file non viene salvato	
Primary Actor		Front-End	
Trigger		Il Framework riceve una richiesta di “upload file”	
Description	Step n°	Actor	System
	1		Riceve la richiesta. Contatta il DRMS per fare lo store del file
	2		Contatta il DBMS per registrare il file

	3		Riceve un O.K. dal DBMS
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un errore
Extensions 2: Errore DRMS	Step n°	Actor	System
	2a		Riceve un K.O. Invia un errore

Use Case # 19		Upload File da URI	
Goal in Contex		Il Front-End vuole fare l'upload di un file	
Scope & Level			
Preconditions			
Success End Conditions		Il file viene caricato con successo. Il Front-End riceve un O.K.	
Failed End Conditions		Il file non viene caricato. Il Front-End riceve un K.O.	
Primary Actor		Front-End	
Trigger		Il Front-End invia una richiesta "upload file da URI"	
Description	Step n°	Actor	System
	1	Il Front-End invia una richiesta "upload file da URI", che include l'URI del	

		file	
	2		Fa lo store del file (riferirsi ad UC #7)
	3		Invia un O.K. al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un errore
Extensions 2: Errore DRMS	Step n°	Actor	System
	2a		Riceve un K.O. Invia un errore

Use Case # 17			Elimina file
Goal in Contex			Il Front-End vuole eliminare un file
Scope & Level			
Preconditions			
Success End Conditions			Il file viene eliminato ed il Front-End riceve un O.K.
Failed End Conditions			Il file non viene eliminato ed il Front-End riceve un K.O.
Primary Actor			Front-End
Trigger			Invia una richiesta “elimina file”
Description	Step n°	Actor	System
	1	Invia una richiesta “elimina	

		file", che include l'URI del file	
	2		Riceve la richiesta e contatta il DBMS per rimuovere il file dal Database
	3		Riceve un O.K. dal DBMS e contatta il DRMS per eliminare il file dal File
	4		Riceve un O.K. dal DRMS e ritorna un O.K. al Front-End
Extensions	Step n°	Actor	System
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un errore
Extensions 2: Errore DRMS	Step n°	Actor	System
	2a		Riceve un K.O. Invia un errore
Extension 3: file non trovato			
	3b		Il file non è stato trovato. Invia un K.O. al Front-End

Use Case # 18	Estrai colonne
Goal in Contex	Il Front-End vuole creare un nuovo file con le colonne selezionate
Scope & Level	
Preconditions	
Success End Conditions	Il File viene creato con successo ed il Front-End riceve il suo URI
Failed End Conditions	Il file non viene creato ed il Front-End riceve un K.O.
Primary Actor	Front-End
Trigger	Il Front-End invia una richiesta “Estrai colonne”

Description	Step n°	Actor	System
	1	Invia una richiesta “Estrai colonne”, che include l’URI del file e le colonne da estrarre	
	2		Riceve la richiesta. Contatta il DRMS per ottenere il file originale
	3		Riceve un O.K. Crea il nuovo fil con le colonne selezionate.
	4		Fa lo store del file (referirsi ad UC#7).

	5		Invia un O.K. al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	4a		Non riesce a contattare il DBMS. Invia un errore
Extensions 2: Errore DRMS	Step n°	Actor	System
	4a		Riceve un K.O. Invia un errore

5.1.4. Esperimenti

Use Case # 12		Controlla stato	
Goal in Contex		Il Front-End vuole conoscere lo stato di un esperimento	
Scope & Level			
Preconditions			
Success End Conditions		Il Front-End riceve lo stato dell'esperimento e le URI dei file di output che sono pronti	
Failed End Conditions		Il Front-End riceve un K.O.	
Primary Actor		Front-End	
Trigger		Il Front-End invia una richiesta "controlla stato"	
Description	Step n°	Actor	System
	1	Invia una richiesta "controlla stato" che	

		include l'id dell'esperimento	
	2		Riceve la richiesta. Contatta il DBMS per ottenere lo stato dell'esperimento e gli URI dei file di output che sono pronti.
	3		Riceve i dati dal DBMS
	4		Contatta il DRMS per ottenere informazioni sullo stato del job
			Riceve le informazioni dal DRMS e le ritorna al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	4a		Non riesce a contattare il DBMS. Invia un errore
Extensions 2: esperimento non trovato	Step n°	Actor	System
	3b		L'esperimento non è stato trovato. Invia un errore al Front-End

Use Case # 13	Lancia esperimento

Goal in Contex	Il Front-End vuole lanciare un nuovo esperimento		
Scope & Level			
Preconditions			
Success End Conditions	L'esperimento viene creato ed eseguito. Il Front-End riceve l'ID del nuovo esperimento		
Failed End Conditions	Il Front-End riceve un K.O.		
Primary Actor	Front-End		
Trigger	Il Front-End invia una richiesta "lancia esperimento"		
Description	Step n°	Actor	System
	1	Invia una richiesta "lancia esperimento", che include i parametri dell'esperimento	
	2		Riceve la richiesta, crea il nuovo esperimento e lo configura
	3		Contatta il DBMS, creando il nuovo esperimento e registrando I files di output con una flag di presenza "false"
	4		Riceve un O.K. dal l DBMS. Contatta il DRMS per lanciare l'esperimento
	5		Riceve il job ID dal DRMS. Registra l'esperimento sul DBMS
	6		Riceve l'id dell'esperimento dal DBMS e lo invia al Front-End
Extensions 1: Errore	Step n°	Actor	System

DBMS			
	4a		Non riesce a contattare il DBMS. Invia un errore
Extensions 2: Errore DRMS	Step n°	Actor	System
	4a		Riceve un K.O. Invia un errore

Use Case # 14			Registra stato
Goal in Contex			Un esprimento vuole comunciare il suo stato
Scope & Level			
Preconditions			
Success End Conditions			Lo stato viene registrato sul DBMS
Failed End Conditions			Lo stato non viene registrato sul DBMS
Primary Actor			DMPlugin
Trigger			IL DMPlugin invia una richiesta “registra stato”
Description	Step n°	Actor	System
	1	Invia una richiesta “registra stato” che include l’id dell’esperimento, uno stato ed una lista di files	
	2		Riceve la richiesta e contatta il DRMS, per ottenere ulteriori informazioni sullo stato
	3		Riceve lo stato dal DRMS

	4		Contatta il DBMS, registrando lo stato dell'esperimento e modificando I files nella lista contenuta inserendo un "true" nel flag di "presenza"
	5		Riceve un O.K. dal DBMS
Extensions 1: Errore DBMS	Step n°	Actor	System
	4a		Non riesce a contattare il DBMS. Invia un errore

Use Case # 19		Elimina esperimento	
Goal in Contex		Il Front-End vuole eliminare un esperimento	
Scope & Level			
Preconditions			
Success End Conditions		L'esperimento viene eliminato	
Failed End Conditions		L'esperimento non viene eliminato	
Primary Actor		Front-End	
Trigger		Il Front-End invia una richiesta "elimina esperimento"	
Description	Step n°	Actor	System
	1	Il Front-End invia una richiesta "elimina	

		esperimento", che include l'id del'esperimento	
	2		Riceve la richiesta e contatta il DBMS per eliminare l'esperimento
	3		Riceve un O.K. dal DBMS e lo ritorna al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	4a		Non riesce a contattare il DBMS. Invia un errore

5.1.5. Amministrazione

Use Case # 15	Installa plugin
Goal in Contex	Un amministratore vuole installare un nuovo plugin nel sistema
Scope & Level	
Preconditions	
Success End Conditions	Il plugin viene installato
Failed End Conditions	Il plugin non viene installato
Primary Actor	Amministratore
Trigger	L'amministratore invia una richiesta "installa plugin"

Description	Step n°	Actor	System
	1	Invia una richiesta “installa plugin”	
	2		Riceve la richiesta e carica il plugin, ottenendo informazioni da
	3		Contatta il DBMS per registrarlo
	4		Riceve un O.K. e lo ritorna
Extensions 1: Errore DBMS	Step n°	Actor	System
	4a		Non riesce a contattare il DBMS. Invia un errore

Use Case # 16	Disinstalla plugin
Goal in Contex	Un amministratore vuole disinstallare un plugin
Scope & Level	
Preconditions	
Success End Conditions	Il plugin viene disinstallato
Failed End Conditions	Il plugin non viene disinstallato
Primary Actor	Amministratore
Trigger	L’amministratore invia una richiesta “disinstalla plugin”

Description	Step n°	Actor	System
	1	Invia una richiesta “disinstalla plugin”	
	2		Riceve la richiesta e contatta il DBMS per disinstallare il plugin
	3		Riceve un O.K. e lo ritorna
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un errore

5.1.6. Utenti

Use Case # 8	Autentica utente
Goal in Contex	Il Front-End vuole autenticare un utente che sta effettuando il log-in
Scope & Level	
Preconditions	
Success End Conditions	Il Front-End riceve una lista delle sessioni e dei files
Failed End Conditions	Il Front-End riceve un K.O.
Primary Actor	Front-End
Trigger	Il Front-End riceve una richiesta “autentica utente”

Description	Step n°	Actor	System

	1	Invia una richiesta “autentica utente”, che include un nome utente ed una password	
	2		Contatta il DBMS per controllare le credenziali dell’utente
	3		Riceve un O.K.
	4		Contatta il DBMS, richiedendo la lista delle sessioni e dei files
	5		Genera un SSID dalla sessione
	6		Invia la lista delle session, dei files e l’SSID al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	3 a o 4a		Non riesce a contattare il DBMS. Invia un errore
Extensions 1: Credenziali non valide	Step n°	Actor	System
	4a		Le credenziali non sono valide. Invia un errore al Front-End

Use Case # 9	Registra utente

Goal in Contex	Il Front-End vuole registrare un nuovo utente
Scope & Level	
Preconditions	
Success End Conditions	L'utente viene registrato
Failed End Conditions	L'utente non viene registrato
Primary Actor	Front-End
Trigger	Il Front-End invia una richiesta "registra utente"

Description	Step n°	Actor	System
	1	Invia una richiesta "registra utente"	
	2		Riceve la richiesta e contatta il DBMS, chiedendo di registrare un nuovo utente
	3		Riceve un O.K. e lo invia al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un errore
Extensions 1: Utente già esistente	Step n°	Actor	System
	3b		L'utente è già registrato. Invia

			un errore al Front-End

Use Case # 10		Conferma registrazione	
Goal in Contex		Il Front-End vuole confermare la registrazione di un utente	
Scope & Level			
Preconditions		L'utente si è già registrato	
Success End Conditions		La registrazione è confermata	
Failed End Conditions		La registrazione non è confermata	
Primary Actor		Front-End	
Trigger		Il Front-End invia una richiesta "conferma registrazione"	
Description	Step n°	Actor	System
	1	Invia una richiesta "conferma registrazione"	
	2		Riceve la richiesta. Contatta il DBMS per abilitare l'utente
	3		Riceve un O.K. dal DBMS
	4		Invia l'O.K. al Front-End
Extensions 1: Errore DBMS	Step n°	Actor	System
	3a		Non riesce a contattare il DBMS. Invia un errore

Extensions 1: Utente già attivo	Step n°	Actor	System
	3b		L'utente è già attivo. Invia un errore al Front-End

5.2. XML

In questa sezione ci sono i template dei files XML utilizzati nel progetto.

5.2.1. Meta-Data

```
<?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>Meta Data Description</DESCRIPTION>
        <#list field as f>
        <INFO name="field" value="${f}" />
        </#list>
 </VOTABLE>
```

5.2.2. Functionality Description

```
<?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>Functionality General Description</DESCRIPTION>
        <INFO name="functionality" value="${name}" />
        <INFO name="documentation" value="${documentation}" />
        <INFO name="version" value="${version}" />
        <INFO name="creationDate" value="${creationDate}" />
        <!--  Only for submission -->
        <INFO name="userid" value="" />
        <INFO name="sessionid" value="" />
        <INFO name="userChoice" value="" />
        <!--  Only for submission -->
        <#list mode as mode>
        <RESOURCE name="${mode.name}">
                <DESCRIPTION>Running Mode Specific Description</DESCRIPTION>
                <INFO name="contextHelp" value="${mode.documentation}" />
                <TABLE>
                        <GROUP name="inputData">
                        <DESCRIPTION>Input Data Description</DESCRIPTION>
                        <#list mode.inputFields as inputFields>
                        <FIELD name="${inputFields.name}" ucd="${inputFields.ucd}"
unit="${inputFields.unit}" datatype="${inputFields.type}"
                        precision="${inputFields.precision}">
                                <DESCRIPTION>${inputFields.description}</DESCRIPTION>
                                <#if inputFields.constraintType="Range">
                                <VALUES ID="fieldDomain">
```

```
                    <MIN value="${inputFields.constraint[0]}" />
                    <MAX value="${inputFields.constraint[1]}" inclusive="yes" />
                </VALUES>
                </#if>
                <#if inputFields.constraintType="Values">
                <VALUES ID="otherFieldDomain">
                    <#list inputFields.constraint as constraint>
                    <OPTION value="constraint"/>
                    </#list>
                </VALUES>
                </#if>
            </FIELD>
            </#list>
            <#list mode.inputFiles as inputFiles>
            <FIELD name="${inputFiles.name}" ucd="meta.ref.uri;meta.file"
datatype="char" arraysize="200" utype="">
            <!--  utype indica il formato del file e deve essere inserito dal FE quando
sottomette il task. -->
                <DESCRIPTION>${inputFiles.description}</DESCRIPTION>
                <VALUES>
                <!--  Nel contesto di un file le options codificano i possibili tag -->
                <!--  Il FE cambia il valore delle options nel numero di colonne
taggate con il
                    relativo TAG. Il Plugin interpreta il value come valore
```

```
dell'argomento name   -->
                        <#list inputFiles.tags as tags>
                            <OPTION name="${tags.name}" value="" />
                        </#list>
                        </VALUES>
                    </FIELD>
                    </#list>
                    </GROUP>
                    <GROUP name="outputData">
                        <DESCRIPTION>Output Data Description</DESCRIPTION>
                        <#list mode.outputFiles as outputFiles>
                        <FIELD name="${outputFiles.name}" datatype="boolean">
                            <DESCRIPTION>${outputFiles.description}</DESCRIPTION>
                        </FIELD>
                        </#list>
                    </GROUP>
                    <GROUP name="partialOutput">
                        <DESCRIPTION>Partial Output Description</DESCRIPTION>
                        <#list mode.partialOutputFiles as partialOutputFiles>
                        <FIELD name="${partialOutputFiles.name}" datatype="boolean">

        <DESCRIPTION>${partialOutputFiles.description}</DESCRIPTION>
                        </FIELD>
                        </#list>
```

```
                    </GROUP>
                    <!--  Only For Submission-->
                    <DATA>
                        <TABLEDATA>
                        </TABLEDATA>
                    </DATA>
                    <!--  End Data (for submission only)   -->
                </TABLE>
        </RESOURCE>
        </#list>
 </VOTABLE>
```

5.2.3. Functionality list

```
<?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>Functionality List Description</DESCRIPTION>
   <#list func as func>
   <RESOURCE name="${func.name}">
    <DESCRIPTION>${func.description}</DESCRIPTION>
    <INFO name="functionality" value="${func.name}" />
    <INFO name="documentation" value="${func.documentation}" />
```

```
<INFO name="version" value="${func.version}" />
<INFO name="creationDate" value="${func.creationDate}" />
<INFO name="domains" value="${func.domains}" />
<TABLE>
  <GROUP name="RunModes">
   <DESCRIPTION>This Functionaly supports different run modes</DESCRIPTION>
   <FIELD datatype="char" name="Train">
     <DESCRIPTION>mode description</DESCRIPTION>
   </FIELD>
   <FIELD datatype="char" name="Test">
     <DESCRIPTION>mode description</DESCRIPTION>
   </FIELD>
   <FIELD datatype="char" name="Run">
     <DESCRIPTION>mode description </DESCRIPTION>
   </FIELD>
   <FIELD datatype="char" name="Full">
     <DESCRIPTION>mode description</DESCRIPTION>
   </FIELD>
  </GROUP>
  <DATA>
   <TABLEDATA>
     <tr><td> ${func.trainvalue} </td> </tr>
     <tr><td> ${func.testvalue} </td> </tr>
     <tr><td> ${func.runvalue} </td> </tr>
```

```
            <tr><td> ${func.fullvalue} </td> </tr>
          </TABLEDATA>
        </DATA>
      </TABLE>
    </RESOURCE>
    </#list>
</VOTABLE>
```

5.2.4. Interactive Report

```
<?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>Functionality General Description</DESCRIPTION>
<INFO name="functionality" value=${func.name} />
<INFO name="documentation" value=${func.documentation}/>
<INFO name="version" value=${func.version} />
<INFO name="creationDate" value=${func.creationDate}/>
<INFO name="user mail" value=${userMail} />
<INFO name="sessionid" value=${sessionId} />
<INFO name="runningMode" value=${runningMode} />
<INFO name="status" value=${status} />
<INFO name="error" value=${error} />
```

```
<INFO name="lastAccessData" value=${lastAccess} />
<INFO name="name" value=${expName} />
<RESOURCE name=${runningMode}>
        <DESCRIPTION>Running Mode Specific Description</DESCRIPTION>
        <TABLE>
                <GROUP name="outputData">
                        <DESCRIPTION>Output Data Description</DESCRIPTION>
                        <#assign i = 1>
                        <#list outputFiles as of>
                        <FIELD name=${of.name} datatype="boolean">
                                <DESCRIPTION>Output ${i} description</DESCRIPTION>
                        </FIELD>
                        <#assign i = i+1>
                        </#list>
                </GROUP>
                <GROUP name="partialOutput">
                        <DESCRIPTION>Partial Output Description</DESCRIPTION>
                  <#assign i = 1>
                        <#list partialOutputFiles as pof>
                        <FIELD name=${pof.name} datatype="boolean">
                                <DESCRIPTION>Partial Output ${i} description</DESCRIPTION>
                        </FIELD>
                  <#assign i = i+1>
                        </#list>
```

```
            </GROUP>
            <DATA>
                <TABLEDATA>
                    <#if (nOutputFiles>0 )>
                    <TR>
                    <#list outputFiles as of>
                        <TD> of.value </TD>
                    </#list>
                    <#list partialOutputFiles as pof>
                        <TD> pof.value </TD>
                    </#list>
                    </TR>
                    </#if>
                </TABLEDATA>
            </DATA>
        </TABLE>
</RESOURCE>
</VOTABLE>
```

5.2.5. Session Manager

```
<?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>Session Management List</DESCRIPTION>
        <INFO name="user mail" value="${userMail}" />
        <INFO name="fullname" value=${name}/>
        <INFO name="country" value="${country}" />
        <INFO name="affiliation" value="${affiliation} />
        <INFO name="creationDate" value=${creationDate}/>
        <#list sessions as s>
        <RESOURCE name=${s.name} ID=${s.id}>
                <DESCRIPTION>A Session</DESCRIPTION>
                <INFO name="creationDate" value=${s.creationDate} />
                <INFO name="lastAccess" value="${s.lastAccess}" />
                <TABLE ID="files">
                        <DESCRIPTION>Session Files Uploaded by the user himself</DESCRIPTION>
                        <#list s.files as f>
                        <GROUP name=${f.name}>
                                <DESCRIPTION>A filename</DESCRIPTION>
                                <PARAM name="creationDate" datatype="char" arraysize="*"
value=${f.creationDate} />
                                <PARAM name="lastModification" datatype="char" arraysize="*"
value=${f.lastMod} />
                                <PARAM name="uri" datatype="char" arraysize="*" value=${f.uri} />
                        </GROUP>
                        </#list>
```

```
            </TABLE>
            <TABLE ID="Experiments">
                <DESCRIPTION>Session Experiments</DESCRIPTION>
                <#list s.experiments as e>
                <GROUP name=${e.name}>
                    <DESCRIPTION>An Experiment</DESCRIPTION>
                    <PARAM name="id" datatype="long" value=${e.id} />
                    <PARAM name="status" datatype="char" arraysize="*"
value=${e.status} />
                    <PARAM name="lastAccess" datatype="char" arraysize="*"
value=${e.lastAccess} />
                    <PARAM name="errorString" datatype="char" arraysize="*"
value=${e.error} />
                    <PARAM name="functionality" datatype="char" arraysize="*"
value=${e.func} />
                    <PARAM name="creationDate" datatype="char" arraysize="*"
value=${e.creationDate} />
                </GROUP>
                </#list>
            </TABLE>
        </RESOURCE>
        </#list>
</VOTABLE>
```

5.2.6. Status Report

```
<?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>Status Report</DESCRIPTION>
        <INFO status="${status}>
        <INFO experimentName="${name}>
        <#list files as f>
        <INFO name="${f.name}" value="${f.uri}" />
        </#list>
  </VOTABLE>
```

5.3. Testing

Esistono due modi principali per effettuare testing di un prodotto. Il primo è la verifica statica, che consiste nell'ispezione manuale del codice sorgente. Questo metodo può essere molto efficace per individuare i bugs, ma richiede molto tempo. Il secondo metodo è l'ispezione dinamica, che consiste nel testare risultati ottenuti nell'esecuzione del prodotto con quelli predetti da un oracolo. Il testing dinamico può essere effettuato a vari livelli, partendo dalle singole unità e testando successivamente il sistema intero. Ci sono due approcci diversi per effettuare questo tipo di testing: black-box e white-box. Nel testing black-box il sistema è trattato come una scatola nera. Questa metodologia di testing è ottima per testare il sistema confrontandolo con il documento dei requisiti ma non può trovare difetti nel codice. La seconda metodologia di testing è la white-box, che consiste invece nel testare il codice delle unità,

forzando i diversi percorsi al suo interno nei vari casi di test.

Dato che la suite non è ancora completamente costruita, per il momento è stato possibile effettuare solo l'unit testing del Framework. Per questo motivo, è stato fondamentale il lavoro di Scaffolding. Lo scaffolding consiste nel costruire moduli che simulino l'ambiente circostante dell'unità. Questi moduli sono detti stubs se sono chiamati dall'unità e driver se simulano l'ambiente chiamante. Le varie unità della suite sono state simulate nei seguenti modi:

- Front-End: le chiamate del Front-End sono state simulate semplicemente con un piccolo client java.
- DRMS: è stato realizzato un semplice driver per lo Stand-Alone, che lavora sul disco rigido sul quale si testa il Framework.
- DBMS: si è simulato un DBMS nella memoria utilizzando una HashMap.

Al momento l'unica strategia di testing che è stata utilizzata è la black-box. E' necessario adottare anche la metodologia white-box per un testing più approfondito del sistema, e possibilmente anche metodologie statiche.

I casi di test trattati per il momento sono contenuti nelle tabelle seguenti:

TEST ID	1
TEST NAME	Verifica IP
TEST DESCRIPTION	Il test ha lo scopo di verificare il funzionamento del meccanismo di sicurezza basato su IP.

INPUT	Desired Input	Obtained Output
Comunicazione da IP autorizzato	Connessione con Successo	Connessione con successo
Comunicazione da IP non autorizzato	Connessione fallita	Connessione fallita
LOCATION		

TEST ID	2	
TEST NAME	File Management	
TEST DESCRIPTION	Questo test ha lo scopo di verificare la correttezza delle procedure di trasferimento dei files	
INPUT	Desired Input	Obtained Output
Upload di un file con successo	File trasferito completamente. Il Front-End riceve l'URI del file	File trasferito completamente. Il Front-End riceve l'URI del file
Upload di un file incompleto su Framework	Il Front-End riceve un errore	Il Front-End riceve un errore
Upload di un file incompleto su FileStore	Il Framework riceve un errore dal DRMS e	Il Framework riceve un errore dal DRMS e

	lo trasferisce al Front-End	lo trasferisce al Front-End
Download di un file da Framework ad utente	Il Front-End riceve il file	Il Front-End riceve il file
Download di un file da Grid a Framework	Il Framework riceve un file	Il Framework riceve un file
Estrazione delle colonne da un file	Il Front-End riceve l'URI del file con le colonne estratte	Il Front-End riceve l'URI del file con le colonne estratte
LOCATION		

TEST ID	3	
TEST NAME	XML Management	
TEST DESCRIPTION	Questo test ha lo scopo di verificare la correttezza del parsing e della generazione dei documenti XML	
INPUT	Desired Input	Obtained Output
Richiesta di file XML:	Documento ben	Documento ben

Functionality Description Functionality List Interactive Report Session Manager Meta-Data Status Report	formato e valido	formato e valido
Parsing di un documento ben formato ottenuto dal Front-End	Parsing con successo	Parsing con successo
Parsing di un documento mal formato ottenuto dal Front-End	Errore nel parsing	Errore nel parsing
LOCATION		

6. Bibliografia

- Progetto S.C.O.P.E. - http://www.scope.unina.it
- Progetto Vo-Neural \ Dame - http://voneural.na.infn.it
- Java - http://java.sun.com
- Restlet: Lightweight REST frame work - http://www.restlet.org
- W3C: World Wide Web Consortium - http://www.w3.org/XML/
- IvoaVOTable - http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/IvoaVOTable
- Freemarker: Java Template Engine Library - http://freemarker.sourceforge.net/