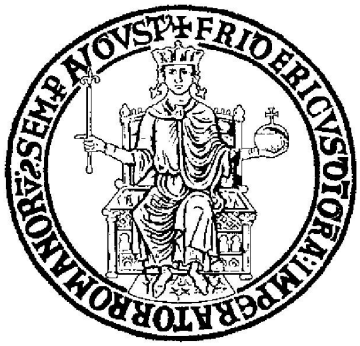
UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II
Facoltà di Scienze
Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica




# Sviluppo dell'applicazione DAMEGRID per il progetto DAME attraverso l'infrastruttura SCoPE-GRID

Tesi Sperimentale di Laurea Triennale

*Tutor Accademico*
Dott. Adriano Peron

*Candidato*
Giovanni Vebber
Matricola 566/346

*Tutor Aziendale*
Dott. Massimo Brescia

ANNO ACCADEMICO 2010/2011

# INDICE GENERALE

# INDICE TABELLE

# INDICE FIGURE

# Introduzione

La presente tesi, descrive li lavoro di tirocinio da me svolto, presso L'Osservatorio Astronomico di Capodimonte, nell'ambito del progetto DAME.
L'attività svolta è stata quella di effettuare un restyling del componente Driver Management System, della suite del progetto DAME. In modo da aggiungere in tale componente, nuove funzionalità che permettono di effettuare operazioni su una infrastruttura GRID (e nello specifico la griglia SCoPE-GRID), con lo scopo di poter usufruire delle capacità di calcolo e archiviazione di cui l'infrastruttura dispone.

L'idea di poter interagire con una infrastruttura GRID, nasce dall'esigenza di voler ottenere una maggiore capacità di calcolo, per l'attività che il progetto DAME svolge, cioè l'estrazione di informazioni significative da grandi quantità di dati (soprattutto di tipo astronomico), attraverso l'attività di data mining (vedi capitolo 1).
Nella versione beta attuale, della suite, tutti gli esperimenti vengono elaborati su macchina stand alone. Tale situazione però non rappresenta la migliore soluzione possibile. Infatti, essendo l'attività di data mining, incentrata sull'estrazione di informazioni su grandi quantità di dati, può capitare che l'elaborazione di un esperimento vada ad impiegare molto tempo per terminare. Quindi il più delle volte si andrebbe a tenere impegnata la singola macchina, che sta elaborando l'esperimento, per un lungo periodo di tempo. Considerando poi che la suite prevede una gestione in multithreading per gli esperimenti, si andrebbe ad appesantire il tutto con un costo computazionale eccessivo. Da queste considerazioni, è nata l'idea di integrare nel componente DRMS della suite, le funzionalità che permettono di interagire con una infrastruttura GRID, in modo da ottenere una maggiore capacità di calcolo e archiviazione dei dati, molto utile per l'attività che la suite DAME svolge.

Come si vedrà in modo dettagliato nei prossimi capitoli, il lavoro svolto durante il tirocinio è stato diviso in due parti.
La prima parte aveva come scopo principale, quello di ottenere un punto d'accesso, cioè una User Interface (che rappresenta l'unico mezzo di comunicazione con una struttura GRID), per utilizzare le risorse di calcolo e archiviazione che una griglia (nel caso specifico SCoPE-GRID), mette a disposizione. Quindi per prima cosa, si è dovuto predisporre una macchina (pc desktop), sulla quale è stato installato il middleweare gLite (nel caso specifico gLite 3.2), riferito al componente UI (User Interface). Inoltre tale macchina doveva anche disporre di particolari requisiti hardware richiesti dal middleweare stesso (vedi capitolo 3). In un secondo momento, una volta che il software era funzionante e con la User Interface disponibile, si è configurata quest'ultima, in modo da poter accedere alle risorse del sito SCoPE-GRID.
La seconda parte del tirocinio è stata incentrata sull'analisi, progettazione e sviluppo del progetto (denominato DAMEGRID), in ambiente Java (vedi capitoli 4-5). Si sono quindi, attraverso le metodologie dell'ingegneria del software, prima individuati i requisiti funzionali, poi attraverso i vari diagrammi (use case, tabelle di cockburn, etc), si è individuata tutta la logica di funzionamento. Finita l'analisi si è passati alla progettazione dove si è identificata l'architettura generale del progetto e poi tramite i diagrammi di classe, di sequenza, si sono definite le classi e le loro interazioni. Finita la progettazione si è passati allo sviluppo vero e proprio, dove si sono principalmente implementate tutte le funzionalità (modulo DriverGrid), che permettono di utilizzare i comandi gLite che interagiscono con l'infrastruttura GRID. Tali comandi sono stati gestiti attraverso delle system call Java (vedi paragrafo 5.6), opportunamente implementate, in modo da garantire una totale automazione dei comandi utilizzati.

I capitoli della presente tesi sono così suddivisi:

Nel capitolo uno è presente una breve introduzione del progetto DAME, con descrizione dei componenti della suite.

Nel capitolo due viene presentato il Grid computing, in particolare viene descritta la sua architettura, le caratteristiche di base, la sicurezza e viene fatta anche una descrizione generale sul middleweare gLite.

Nel capitolo tre viene descritta l'attività su come installare e configurare una User Interface (attraverso il middleware gLite versione 3.2), per ottenere un punto d'accesso alla griglia SCoPE.

Nel capitolo quattro si descrive la fase di analisi (use case, cockburn, etc), e definizione dei requisiti che il progetto deve soddisfare.

Nel capitolo cinque è presente la progettazione e le specifiche tecniche utilizzate. Inoltre viene fornita anche una breve discussione sull'attività di testing.

# 1 Il Progetto DAME

Dame è un progetto che nasce all'inizio del 2007 (in origine il nome era VO-Neural), e consiste nello sviluppo di suite di data mining orientate al web per la ricerca scientifica astronomica, conforme agli standard internazionali istituiti dall'**International Virtual Observatory Alliance** (IVOA). Il data mining rappresenta un processo automatizzato attraverso il quale è possibile estrarre informazioni significative da grandi quantità di dati, mediante tecniche e algoritmi specifici. Ai giorni nostri reperire e depositare informazioni (dati), è pratica semplice. Molto meno semplice è la capacità di utilizzare, queste grandi quantità di dati, che possono essere molto eterogenei tra loro, non strutturati, ridondanti, in breve senza l'impiego del data mining sarebbe notevolmente complicato se non impossibile estrarre informazioni. Da quanto detto si comprende meglio il concetto di data mining, che non ricopre solo un ruolo utile nelle attività scientifiche ma anche in altri settori. Il progetto DAME[1] è finanziato dalla comunità europea e dal MIUR[2], e comprende molte collaborazioni con altri enti:

- S.C.o.P.E;
- International Virtual Observatory Alliance (IVOA);
- The European Virtual Observatory (EURO-VO);
- Italian Ministry of Research (MIUR);
- Dipartimento di Ingegneria Informatica Università degli Studi di Napoli Federico II;
- Sezione Informatica Dipartimento Scienze Fisiche - Università degli Studi di Napoli Federico II;
- Virtual Observatory Technological Infrastructures (VOTECH);
- INAF - Osservatorio Astronomico di Trieste (VO-AIDA).

Si considera nel seguito la composizione.

## 1.1. La suite DAME

L'applicazione DAME (o DAMEWEARE), è composta da alcuni componenti che interagiscono strettamente per svolgere operazioni (vedi figura 1).

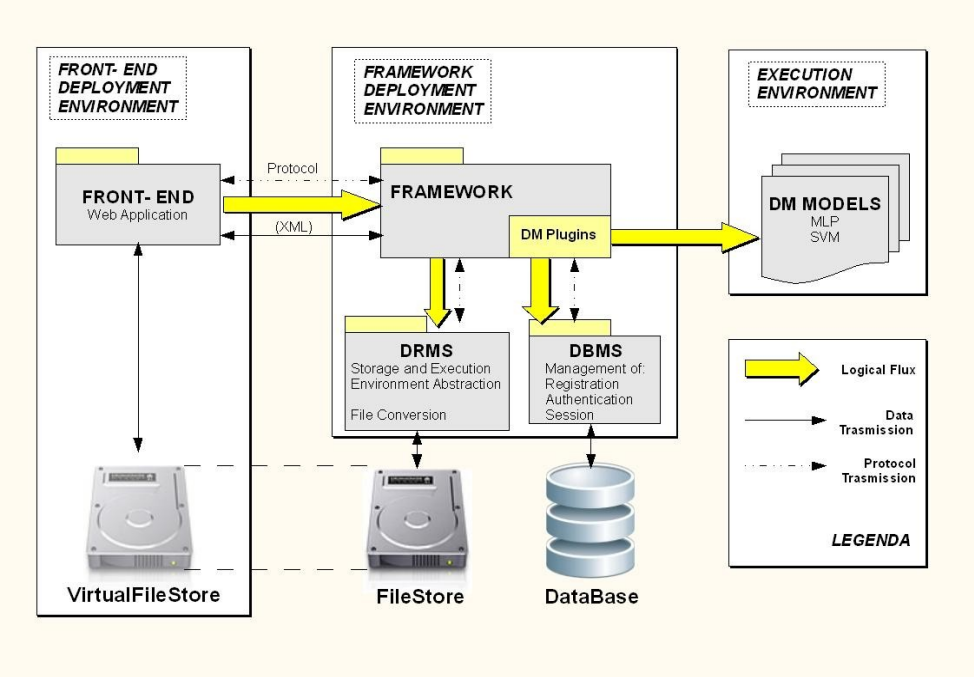



**Figura 1: Flusso funzionale tra componenti**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

[1]DAME: DAta Mining & Exploration
[2]MIUR: Italian Ministry of Research

I componenti sono:

- **Front-End (FE)**: front end della web-application, composta da una GUI (realizzata con tecnologia GWT) e da un sistema client/server (basato su Java servlet), responsabile della gestione delle interazioni tra l'utente finale ed il sistema interno;
- **Database Management System (REDB)**: una libreria di classi, responsabile della gestione delle interazioni con la base dati (informazioni utente ed I/O degli esperimenti);
- **Driver Management System (DRMS)**: una libreria di classi che ha il compito di astrarre l'applicazione rispetto all'infrastruttura hardware sottostante (GRID, CLOUD, Stand alone);
- **Framework (FW)**: il cuore dell'applicazione, un componente basato su un ambiente privo di stato (restful stateless), con il compito di gestire il flusso di comandi/dati tra tutti gli altri componenti;
- **Data Mining Models (DMM)**: modulo comprendente le librerie software (sottoforma di API), che implementano i vari algoritmi di data mining integrati nella web application;
- **DMPlugin**: sub-componente (tra FW e DMM), responsabile dell'elaborazione e setup degli esperimenti utente, in cui viene implementata l'associazione funzionalità-modello per ogni esperimento (attraverso l'interpretazione dei parametri scelti) e la relativa esecuzione con salvataggio dell'output e notifica all'utente dello stato dell'esperimento.

### 1.1.1 Framework (FW)

Il componente Framework rappresenta il nucleo della suite DAME, principalmente ha il compito di gestire l'attività di comunicazione tra il Front End (quindi l'utente finale), e le altre componenti. Questa interazione tra il Framework e il Front End, permette ad un qualsiasi utente di potersi registrare, lanciare e configurare esperimenti (nella sua sessione lavorativa), e di visualizzare risultati. In pratica si permette all'utente di poter interagire con la suite. Ricapitolando i principali compiti del Framework sono:

- stare in ascolto (e rispondere), alle richieste che arrivano dal Front End e interagire con le altre componenti;
- fornire una interfaccia, utilizzabile da un amministratore, per installare altre funzionalità o effettuare altre attività sul sistema.

L'architettura del Framework si basa su un ***Restful Web Service***, che rappresenta una particolare architettura client-server, dove l'ambiente risulta essere privo di stato e quindi ogni operazione risulta essere atomica. Inoltre le risorse disponibili sono accessibili da un URL[1]. Quindi un client che vuole impiegare una risorsa, sfruttando il classici protocolli HTTP, richiama l'URL che identifica la risorsa desiderata. Il componente Framework, risulta diviso in tre macro elementi principali:

- ***Web Service***: rappresenta l'interfaccia implementata da servlet (ambiente Java), che permette di interagire con il Front End;
- ***Admin Interface:*** interfaccia utilizzata dall'amministratore per svolgere attività nel sistema;
- ***Data Mining Plugin (DMPlugin)***: identificano le funzionalità necessarie di data mining per effettuare un esperimento.

Oltre ai tre elementi sopra elencati, il Framework si compone anche delle seguenti parti:

- ***XMLGenerator***: generatore di file xml utilizzati per la comunicazione con il Front End;
- ***Data Types***: identificano le entità di base della suite;
- ***Error***: rappresenta il pacchetto (package), che gestisce le politiche di errore della suite;
- ***Security***: rappresenta il pacchetto che gestisce la sicurezza della suite.

La figura che segue, mostra complessivamente l'architettura del componente Framework.

[1]URL: Uniform Resource Locator

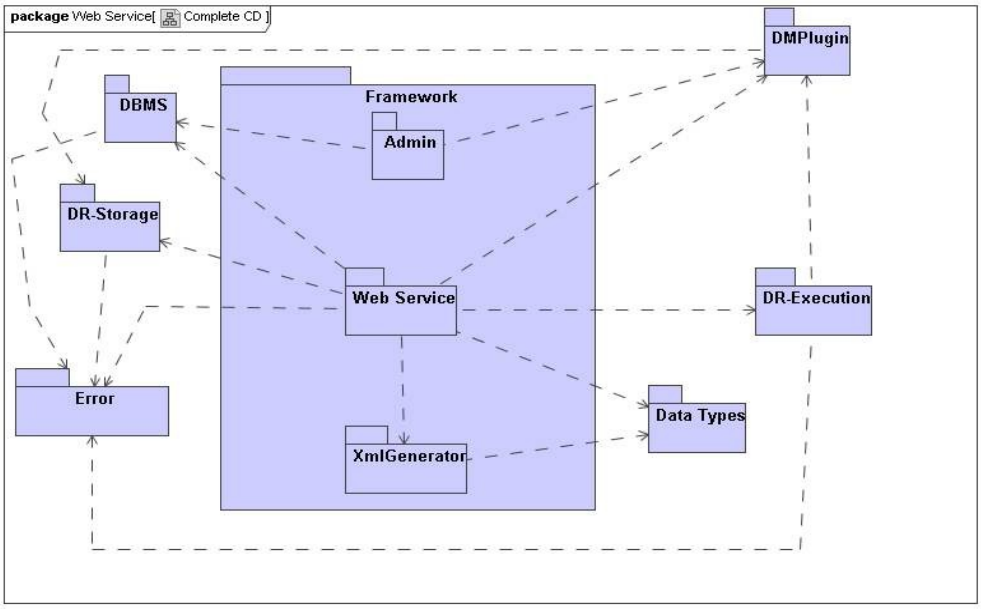



**Figura 2: Architettura componente Framework**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

### 1.1.2 Database Management System (DBMS) & REDB

Il DBMS rappresenta il gestore delle attività di mantenimento delle informazioni manipolate dalla suite DAME. Infatti, il Framework non dispone di capacità di deposito, ma è solamente in grado di reperire, modificare e aggiungere informazioni. E' compito del componente DBMS (attraverso l'interfaccia REDB[1]), svolgere le attività di deposito. Le informazioni gestite dal DBMS sono:

- informazioni sulla registrazione degli utenti;
- informazioni sulle sessioni di lavoro e sugli esperimenti associati;
- informazioni sui dati di input/output;
- informazioni sui dati parziali/finali ricavati dall'esecuzione degli esperimenti lanciati dagli utenti.

Il componente risulta diviso un due parti (vedi figura 3):

- ***REDB***: rappresenta l'interfaccia attraverso la quale è possibile svolgere attività di lettura scrittura sul database;
- ***DBMS***: rappresenta il contenitore dei dati, realizzato sulla base del modello entità-relazioni (ER).

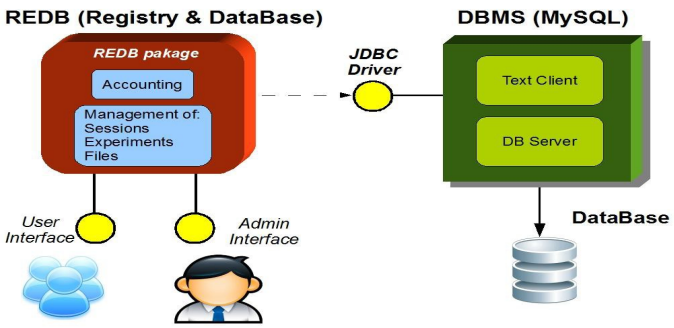



**Figura 3: REDB e DBMS**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

[1]REDB: Registry & Database

Le figure 4 e 5 qui di seguito mostrate descrivono rispettivamente il class diagram REDB e il diagramma entità-relazioni (ER):

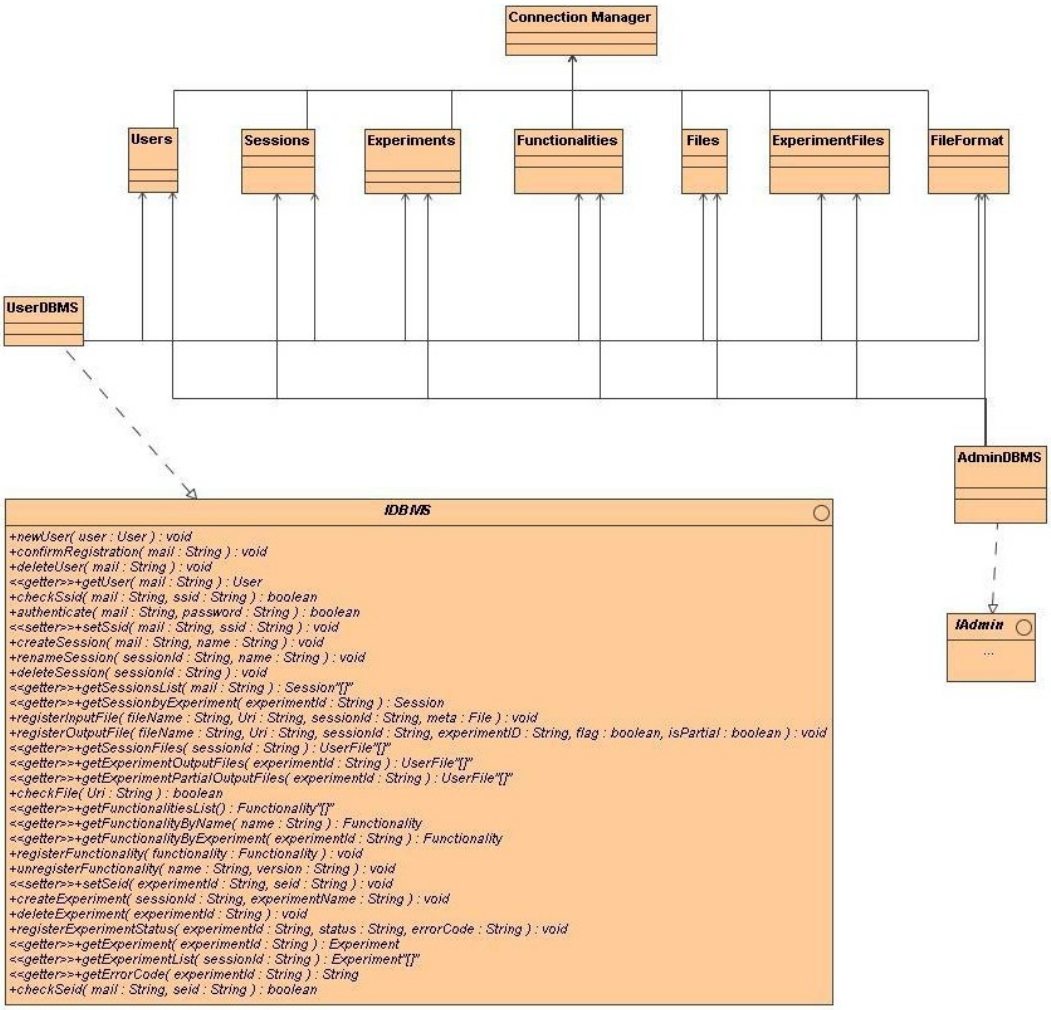



**Figura 4: Diagramma REDB**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

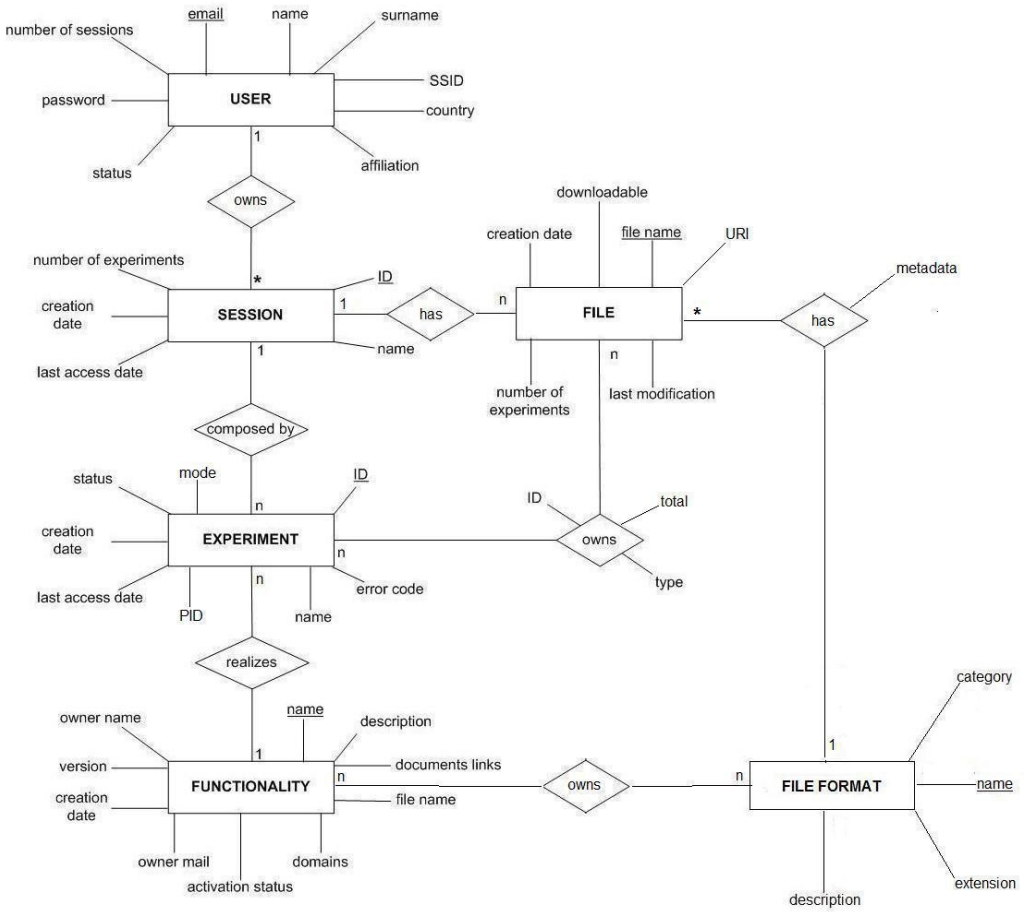



**Figura 5: Diagramma ER**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

### 1.1.3 Data Mining Model (DMM)

Il DMM rappresenta il componente (vedi figura 6), che implementa i modelli di data mining previsti dalla suite DAME. Solitamente i modelli sono scritti mediante diversi linguaggi di programmazione e con paradigmi differenti. Il DMM si basa sulle caratteristiche seguenti:

- Gestione implementativa orientata alle attività, come ad esempio Regressione e Classificazione;
- Disporre di una interfaccia comune tra i modelli, attraverso una classe specifica, in modo da rende i parametri di data mining auto-adattativi;
- Differenziazione tra modelli supervisionati e non supervisionati;
- Possibilità di impiegare su più di un modello la stessa funzionalità, senza duplicazione.

Alcuni modelli implementati dal DMM sono, il Multi Layer Perceptron (MLP), e Support Vector Machine (SVM). Comunque il componente DMM risulta in uno stato "*open*", cioè sempre predisposto a contenere nuovi modelli di data mining.

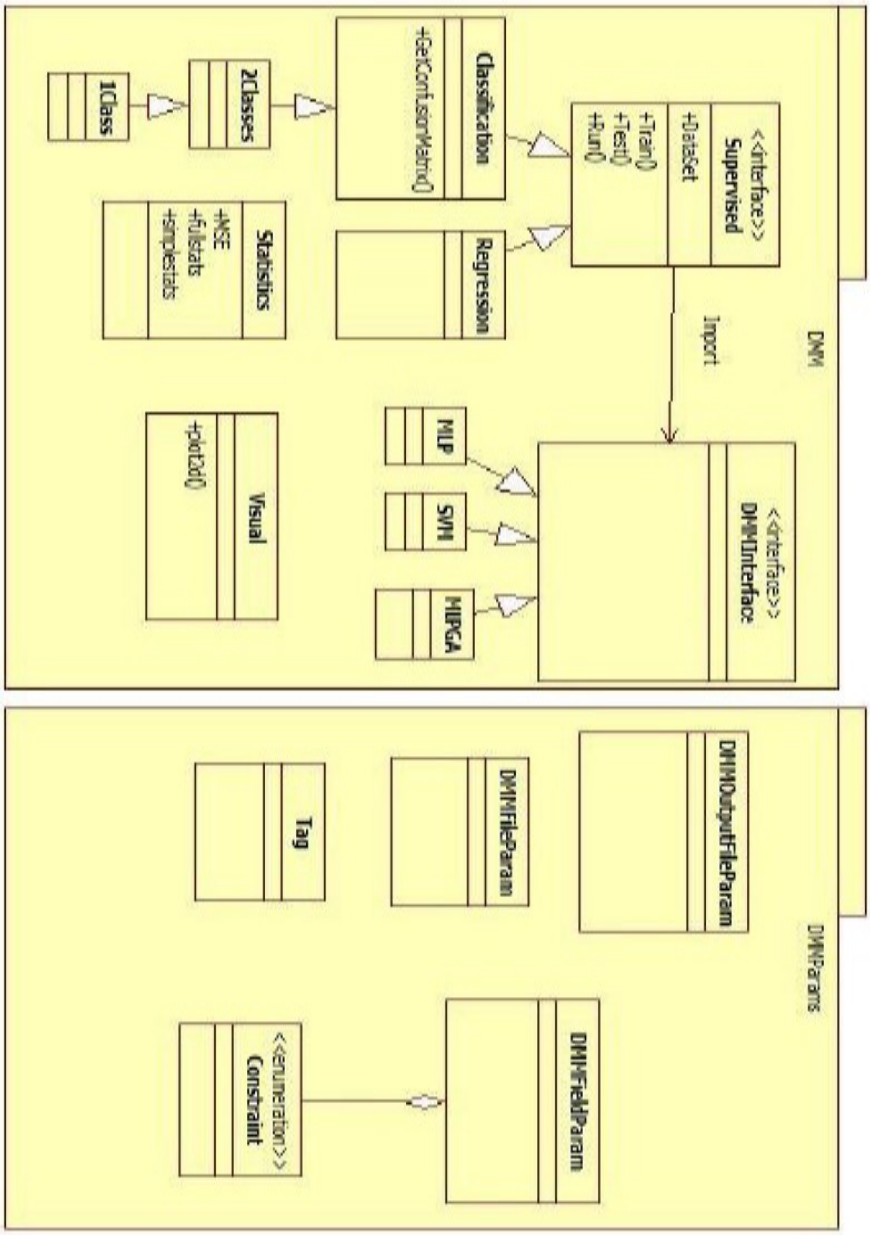



**Figura 6: Architettura Data Mining Model**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

### 1.1.4 Front End (FE)

Il Front End rappresenta il componente, che da un lato interagisce con l'utente e dall'altro comunica con il Framework (figura 7). Il Front End composto da una GUI realizzata con tecnologia GWT (Google Web Toolkit), e da un sistema client/server (basato su Java servlet), fornisce all'utente funzionalità come, la registrazione al sistema di un utente, l'accesso al sistema, il trasferimento dei dati verso il Framework, l'elaborazione dei dati di input, la gestione del workspace (contenitore degli esperimenti), e la visualizzazione dei risultati ottenuti.


**Figura 7: Interazione tra Front End e Framework**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

Il Framework impiega un'architettura di tipo Three-Tier (vedi figura 8), che divide il sistema in tre moduli, dove ogni modulo si dedica ad una sola delle seguenti funzionalità:

- Interfaccia utente;
- Logica funzionale (business logic);
- Gestione dei dati.

I tre moduli interagiscono tra loro attraverso il modello client-server e tramite delle interfacce.

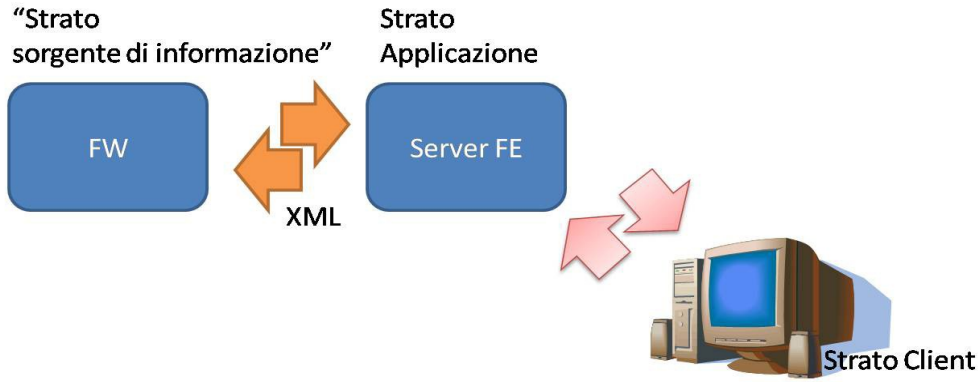



**Figura 8: Architettura FE**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

Il Front End, come già precedentemente detto, è diviso in un lato server e in un lato client. Il lato server è così strutturato:

- due librerie (high level, low level), realizzate in ambiente Java che gestiscono le attività di comunicazione con il Framework;
- dai servizi implementati, basati sulle RPC[1] (figura 9);
- dalle classi che permettono di gestire l'attività di parsing e generazione dei file XML (figura 10).

[1]RPC: Remote Procedure Call

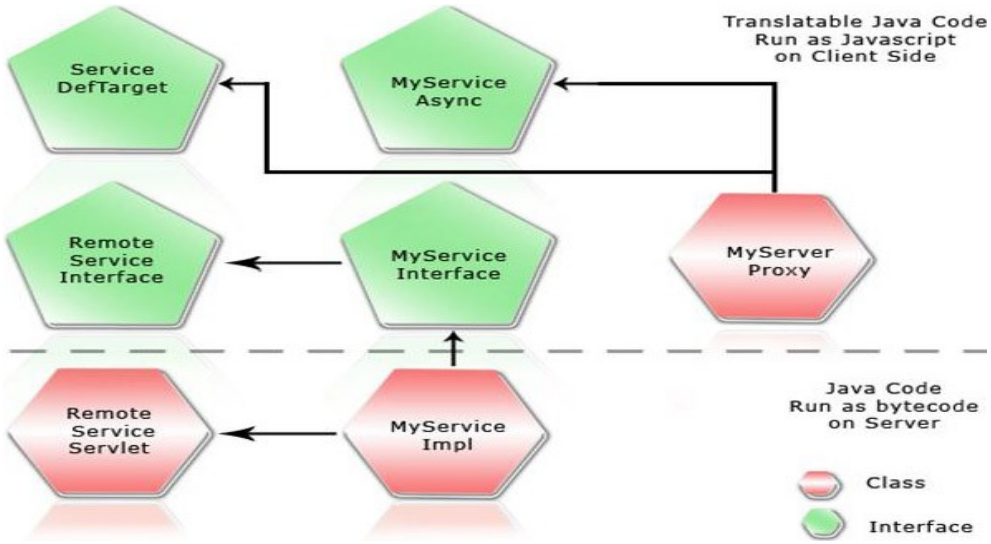



**Figura 9: Modello RPC**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

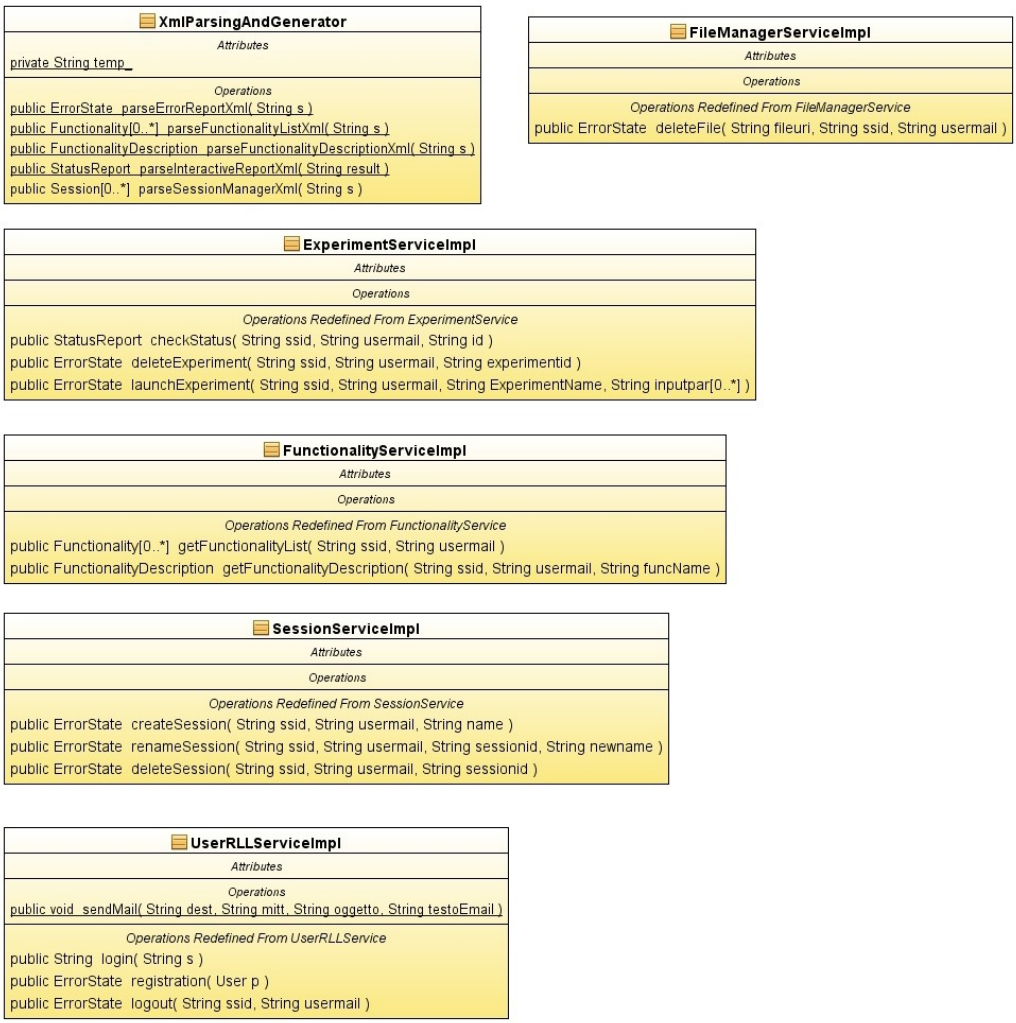



**Figura 10: Generator e Parsing XML**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

Il lato client invece è così composto:

- una GUI, che rappresenta l'interfaccia grafica di interazione con il sistema, utilizzata dall'utente;
- classi che gestiscono le chiamate alle RPC (figura 11), che permettono di inviare e ricevere oggetti Java.

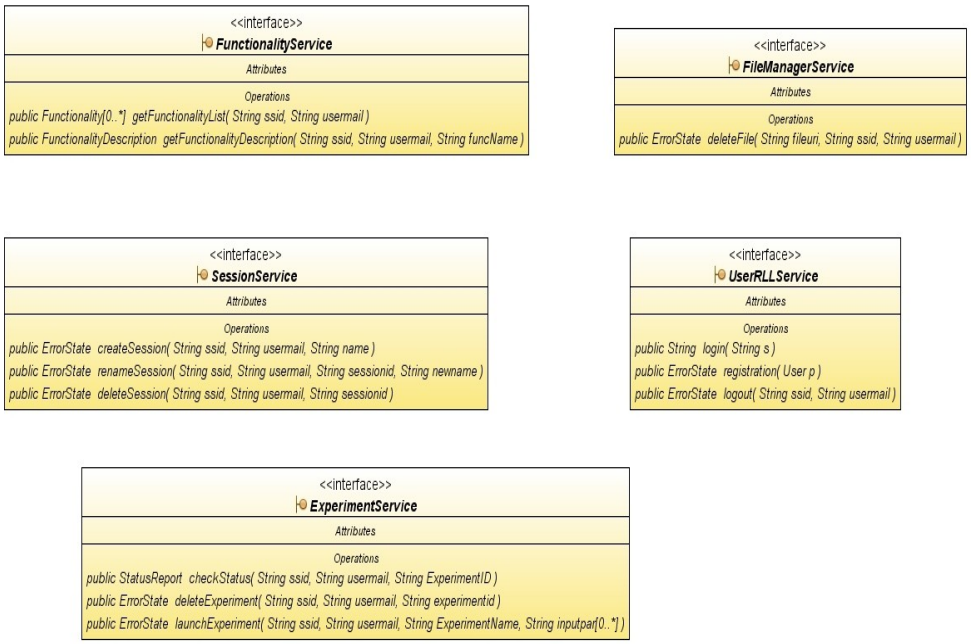



**Figura 11: Servizi RPC**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

### 1.1.5 Driver Management System (DRMS)

Il DRMS rappresenta il componente che il Framework utilizza per svolgere attività di:

- esecuzione di esperimenti sulla piattaforma indicata, che può essere la macchina stand alone o un ambiente GRID;
- archiviazione dei file, attraverso operazioni di download, copia, upload ed eliminazione;
- traduzione dei file nel giusto formato, richiesto dal modello scelto all'interno del componente DMM.

Lo schema (figura 12), qui sotto presentato riassume le attività del DRMS:

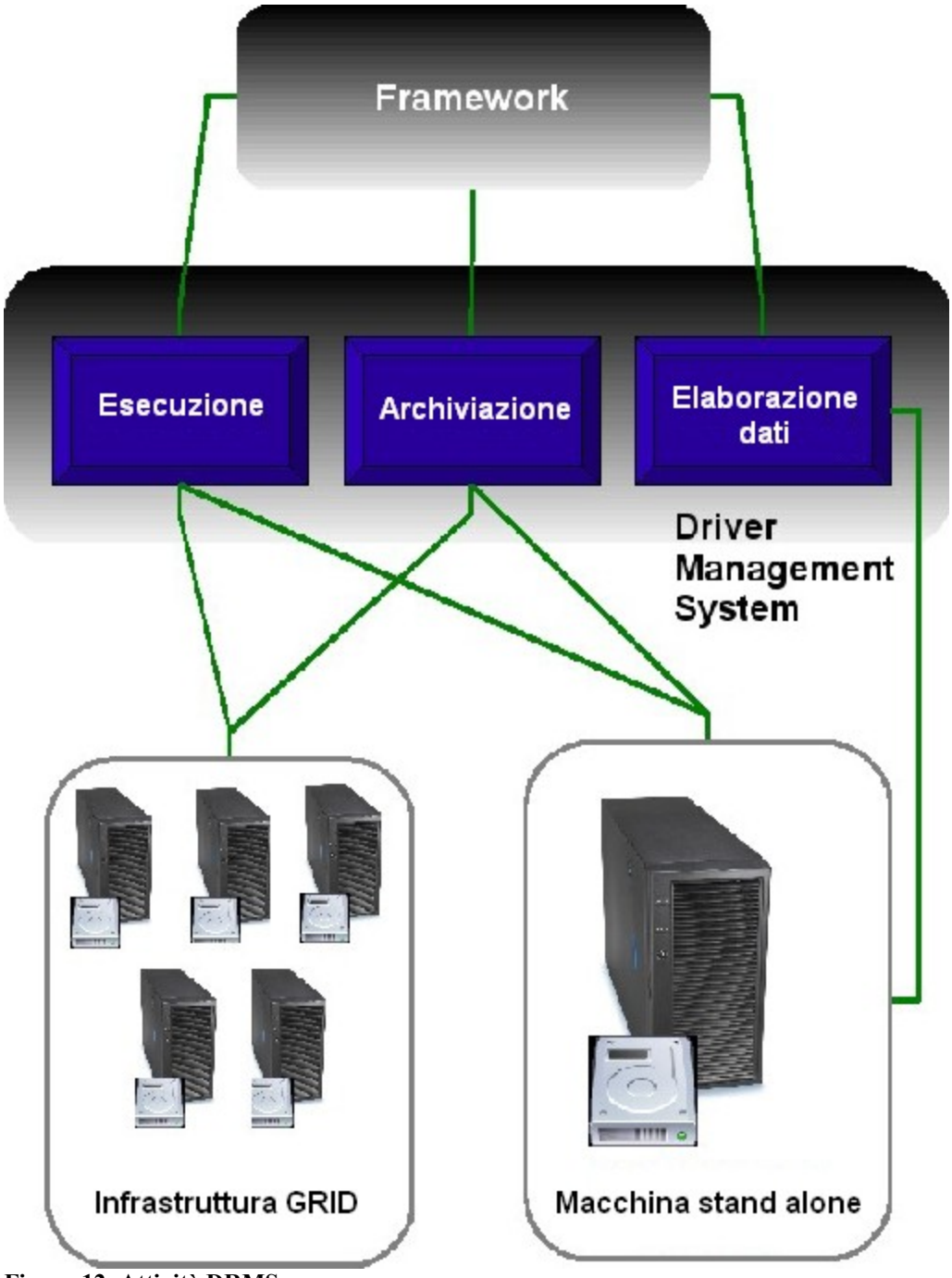



**Figura 12: Attività DRMS**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

Essendo tale componente, uno degli argomenti principali di questa tesi, si rimanda ai prossimi capitoli, per una più approfondita spiegazione delle sue funzionalità e delle modifiche che gli sono state apportate.

# 2 Il Grid computing

Con il termine Grid computing (o sistema Grid), ci si riferisce alla combinazione di più risorse computazionali distribuite su più domini, anche geograficamente distanti, finalizzate per la risoluzione di un obiettivo comune.
Il concetto Grid computing nasce intorno agli anni 90, prendendo poi maggiore considerazione, negli anni successivi, attraverso dei progetti finanziati dalla comunità europea. Lo scopo di tali progetti è stato quello di sviluppare una tecnologia informatica in grado di condividere risorse di calcolo geograficamente distribuite a basso costo che fornissero una grande capacità di elaborazione. I progetti DataGrid e Enabling Grid for E-science in Europe (EGEE), finanziati dalla comunità europea, hanno dimostrato la grande utilità a cui il Grid computing può portare, in termini di sviluppo futuro. Come detto in precedenza, l'obiettivo del Grid computing è quello di fornire, all'utente che vuole sfruttarne le risorse, un sistema di interfacciamento ad alto livello, nascondendo come realmente vengono recuperate le risorse. Il sistema Grid avrà il compito di gestire le richieste che arrivano, di monitorarle e fornire i risultati una volta ottenuti. Tale infrastruttura di basa sia su una componete hardware che su una software.
L'hardware prevede un certo numero di sistemi dotati di potenza di calcolo (cpu), risorse di archiviazione (hard disk), e meccanismi di interconnessione. In realtà essendo l'infrastruttura capace di gestire più sistemi come un unico supercomputer, si può utilizzare hardware a basso costo, favorendo quindi anche un contenimento dei costi realizzativi. Il sistema operativo che gestisce localmente i sistemi è di solito Linux, oppure delle varianti di Unix.
Il componente software rappresenta l'attività su cui maggiormente si sono concentrati gli sviluppi implementativi. In particolare, un consorzio internazionale Globus (guidato da Ian Foster), negli anni ha sviluppato un kit di base per la gestione delle griglie. Tale kit, conosciuto come Globus Toolkit, permette la gestione delle risorse, dei dati, della sicurezza, in pratica di tutto quello che è necessario per gestire in modo efficiente un sistema Grid. Con gli anni, essendo il Globus Toolkit open-source, si sono sviluppati altri software che permettono la gestione di una griglia.

## 2.1. Architettura Grid

Come detto in precedenza il sistema Grid si basa su un concetto di virtualizzazione, cioè permette di collegare più sistemi distribuiti geograficamente, in maniera trasparente a chi l'utilizza. In pratica la virtualizzazione nasconde le complessità di gestione del sistema, dando a chi ne impiega le risorse, l'idea di sfruttare un unico calcolatore molto potente. A livello architetturale le griglie si presentano scomposte in strati, dove ogni strato rende disponibili le proprie funzionalità, agli strati superiori attraverso le interfacce. Tale stratificazione da una parte presenta una maggiore efficienza in termini di protezione, manutenibilità e portabilità, ma presenta anche dei particolari svantaggi. Tali svantaggi sono soprattutto racchiusi nella modifica di una interfaccia di un livello (strato), che comporta la modifica di tutti gli strati che sfruttano quel livello. La figura 13 qui presentata, raffigura l'architettura di base di una infrastruttura Grid.

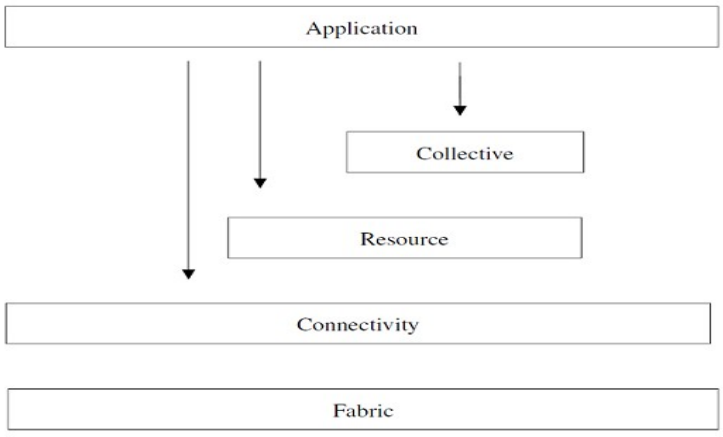



**Figura 13: Architettura Grid**

**[RIF.] Sito SCoPE: http://www.scope.unina.it/default.aspx**

Dalla figura si evince che ogni livello utilizza i servizi offerti dai livelli inferiori. Un ruolo di rilevante importanza nell'architettura Grid è ricoperto dall'interoperabilità, cioè l'insieme dei protocolli comuni, che permettono di negoziare, sfruttare e gestire la condivisione delle risorse. In pratica questi protocolli permettono di indicare agli elementi, che fanno parte di un sistema distribuito, come devono comunicare tra loro. Come visto dalla figura precedente i livelli di un'architettura Grid sono cinque, ognuno con un ruolo ben definito:

- **Fabric**: rappresenta il livello delle risorse fisiche, comprende le risorse di calcolo (cioè richieste per monitorare ed eseguire processi), di archiviazione (salvataggio e recupero file), e di rete (trasferimento dati).
- **Connectivity**: livello che comprende i protocolli di autenticazione e comunicazione per Grid. I protocolli di comunicazione comprendono tutti i protocolli appartenenti al modello iso/osi, mentre quelli di autenticazione forniscono meccaniche di sicurezza, verificando l'identità dell'utente.
- **Resource**: livello che permette di gestire la singola risorsa, permettendone il monitoraggio e il recupero. In pratica sono i protocolli per la gestione della risorse viste singolarmente.
- **Collective**: livello che gestisce l'interazione tra le singole risorse, vedendole come una collezione. I protocolli di questo livello gestiscono le interazioni tra le risorse.
- **Application**: livello che comprende le applicazioni utente che sfruttano i servizi Grid.

L'architettura Grid può essere vista come una clessidra (vedi figura 14), dove il collo di bottiglia viene rappresentato dai livelli Resource e Connectivity. Di solito tale struttura viene denominata middleware. Negli anni sono stati sviluppati molti Grid middleware, come ad esempio DataGrid, Globus, Unicore, etc.

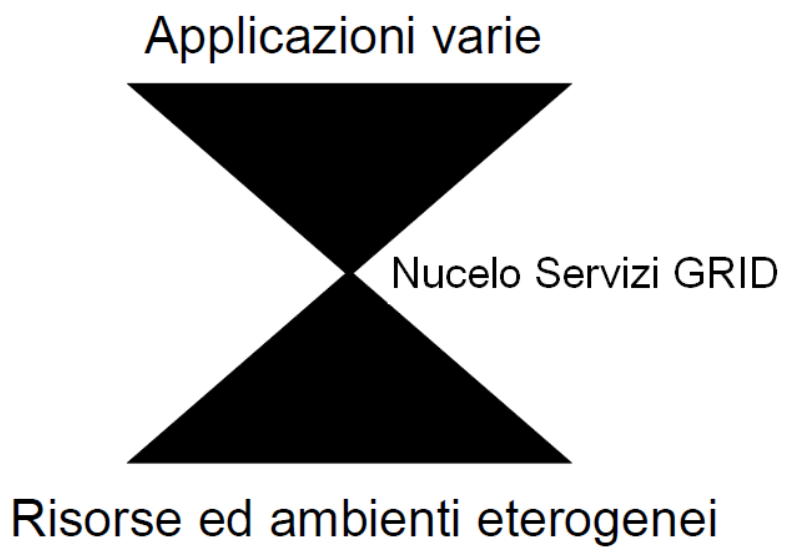



**Figura 14: Schema a clessidra Grid**

Si è già visto che con il termine Grid computing ci si riferisce ad un insieme di entità che condividono risorse geograficamente distribuite. In realtà queste entità possono essere relate tra loro attraverso una Virtual Organization (VO), che rappresenta un insieme dinamico di entità che condividono delle risorse. Le Virtual Organization possono variare per dimensione, scopo e durata. Un requisito fondamentale per un qualsiasi utente che vuole accedere alle risorse Grid, è quello di dover appartenere ad una VO collegata alla griglia di cui si vogliono utilizzare le risorse. La figura 15 rappresenta uno schema di Virtual Organization.

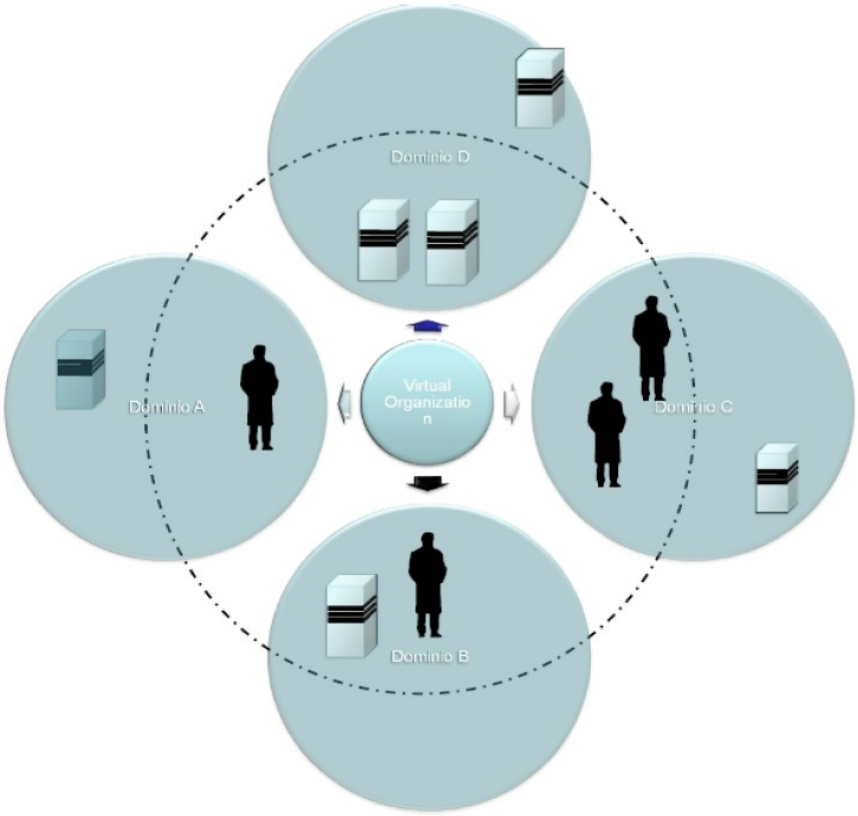



**Figura 15: Virtual Organization**

## 2.2. Sicurezza e autenticazione su Grid

La sicurezza (gestita dalla Grid Security Infrastructure), rappresenta un fattore fondamentale per una infrastruttura Grid, vincolando l'accesso e la condivisione delle risorse attraverso l'utilizzo di certificati X.509.
Un certificato X.509 rappresenta in pratica un documento d'identità digitale, composto da una coppia di chiavi asimmetriche, una pubblica leggibile da tutti, ed una privata gestibile solo dal possessore. Un certificato di questo tipo contiene informazioni come le seguenti:

- Nome del proprietario;
- Chiave pubblica;
- Scadenza del certificato;
- Nome della Certification Authority (CA), che l'ha rilasciato;
- Firma digitale della CA.

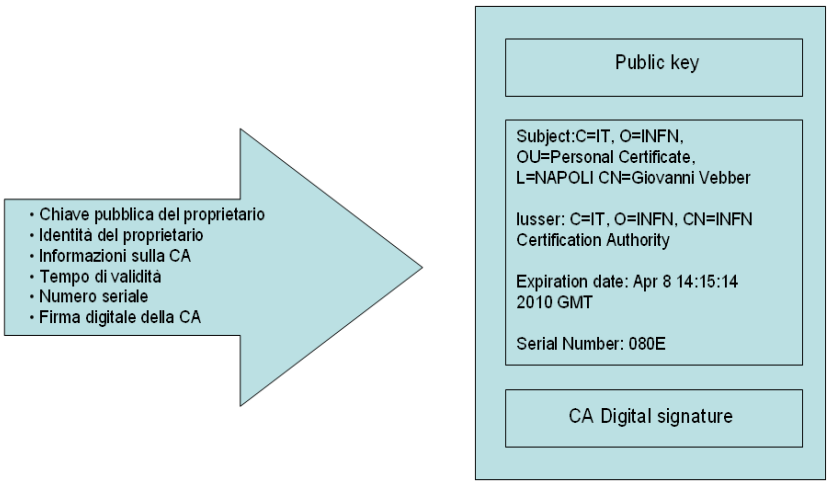



**Figura 16: Certificato X.509**

L'uso del solo certificato digitale comporterebbe però un notevole disagio per l'utente, che durante una sessione lavorativa, sarebbe costretto costantemente ad autenticarsi per ogni attività che vuole svolgere sulla griglia. Ecco perché è stato introdotto un secondo tipo di certificato denominato *proxy*. Un proxy rappresenta un nuovo certificato che ogni utente della griglia può generare attraverso un comando. A differenza del certificato originale, il proxy non è firmato dalla CA ma bensì dall'utente stesso che lo crea (mediante quello originale), inserendo durante la fase di creazione la passphrase[1] della chiava privata in modo da permettere al sistema di poter generare la firma per il nuovo certificato. Il proxy ha una durata limitata nel tempo, di solito 12 ore, dopodiché tale certificato scade e non è possibile più svolgere alcuna attività sulla griglia. Nel periodo in cui il certificato proxy è attivo l'utente può svolgere ogni operazione sulla griglia senza la costante richiesta di autenticazione.
Da quanto detto si comprende che l'uso del proxy è fondamentale quando si vuole lavorare sulla griglia. Tuttavia anche il certificato proxy presenta della problematiche soprattutto dovute alla sua durata limitata, che può comportare la terminazione anticipata di una sessione lavorativa sulla griglia, per un utente che ne stava sfruttando le risorse. Per ovviare in parte a questo problema sono stati introdotti i myproxy, gestiti da server dedicati. Il myproxy server è un servizio offerto dalla griglia che permette di semplificare per l'utente la gestione dei certificati. In breve un utente può generare attraverso un comando un particolare certificato proxy, detto myproxy, che viene conservato nel myproxy server disponibile. In questo modo anche se il certificato proxy scade non si corre il rischio di perdere il lavoro svolto sulla griglia, perché sarà il myproxy stesso a generare autonomamente nuovi certificati per l'utente. La durata del myproxy è superiore a quella di un certificato proxy generico, tipicamente è di sette giorni.

## 2.3. gLite middleware

Il middleware rappresenta un componente software dei sistemi Grid, che permette di rendere trasparente, a chi utilizzata la griglia, la sua complessità. Non esistendo uno standard unico, negli anni sono stati prodotti molti middleware. Uno su tutti è il software gLite, che, anche se non standardizzato è quello più comunemente utilizzato quando si devono gestire infrastrutture Grid. gLite sviluppato dalla European DataGrid, nasce come ambiente software per lo sviluppo di applicazioni commerciali e scientifiche, dove occorre gestire una grande quantità di dati. Fornisce protocolli che permettono l'accesso alle risorse di archiviazione, computazionali e per la creazione di virtual instrument. Il modello gLite si basa sul concetto di “elemento” e “sito”. Dove per elemento si intende un host che fornisce servizi (di tipo collective o core), e metodi per accedere a tali servizi. Inoltre è in grado di pubblicare informazioni sullo stato dei servizi e di interagire con altri elementi Grid o direttamente con gli utenti. Con il termine sito (vedi figura 17), invece si intende, un insieme di risorse che dispongono di almeno un CE (Computing Element), un SE (Storage Element), un gruppo di WN (Worker Node), e di un Site BDII[2]. Ricapitolando il modello gLite implementa due classi di servizi, che sono:

- **Servizi collective**: rappresentano i servizi distribuiti o centralizzati e lavorano ad un livello superiore rispetto alle risorse locali.
  - Virtual Organization Membership Server (VOMS): Servizio per la gestione delle virtual organization e per la gestione delle politiche sulle risorse.
  - Workerload Managment System (WMS): dispone di una visione globale della Gride gestisce(schedula) l'esecuzione dei job sui vari siti.
  - Logging And Bookkeeping Server (LB): tiene traccia e verifica lo stato dei job.
  - Top Bdii (TBDII): servizio che pubblica informazioni di tutte le risorse della Grid e viene usato dalla WMS.
  - Logical File Catalog (LFC): Fornisce servizi di naming (scelta nomi), per un filesystem virtuale usato per mappare file logici su file fisici distribuiti sugli Storage Element.

- **Servizi core**: rappresentano i servizi locali usati per la condivisione delle risorse di calcolo, di archiviazione (storage), e virtual instruments.
  - User Interface (UI): rappresenta un punto d'accesso di un client (utente), verso la griglia. Tramite la UI avviene anche l'autenticazione da parte dell'utente (attraverso il certificato personale), e solo in caso di verifica positiva, è possibile iniziare ad utilizzare le funzionalità e servizi offerti dal middleware.

[1] Passphrase: password definita dall'utente all'atto della creazione del certificato personale
[2] BDII: Berkeley Database Information Index

- Computing Element (CE): usato principalmente per accedere alle risorse computazionali ne fornisce tutti i servizi necessari (principalmente quelli per la gestione dei job). Il CE comunica direttamente con i vari worker node dove vengono effettivamente svolte le attività di calcolo.
- Storage Element (SE): fornisce servizi ed interfacce che permettono di gestire l'archiviazione (storage), locale di un sito Grid in modo trasparente all'utente.
- Worker Node (WN): fornisce servizi di calcolo, rappresenta il blocco delle macchine utilizzate per la vera computazione.
- Site BDII (SBDII): permette il recupero di informazioni sulle risorse locali, utili per effettuare il monitoraggio e la raccolta di informazioni.

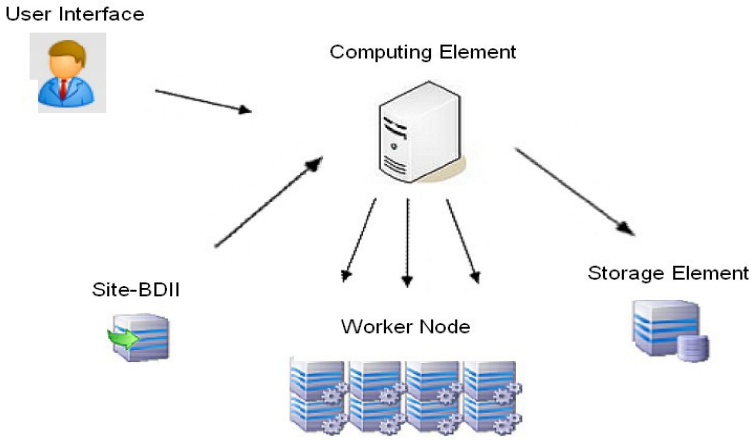



**Figura 17: Sito Grid gLite**

**[Rif.] Sito SCoPE: http://www.scope.unina.it/default.aspx**

## 2.4. Metodi di accesso ad un sito Grid gLite

Sui sistemi GRID, basati sul modulo gLite, l'unico punto di accesso fornito all'utente per utilizzare risorse disponibili, avviene attraverso la User Interface. Quindi la UI rappresenta l'unico ponte di comunicazione con gli altri moduli del middleware gLite. Le funzionalità di base della UI sono le seguenti:

- inoltro di job[1];
- visualizzazione di stato di un job;
- cancellazione di un job;
- copia di un job;
- recupero dell'output di un job;
- ottenere una lista delle risorse disponibili per eseguire job.

Una User Interface può essere utilizzata in due modi (vedi figura 18):

- Tramite un PC desktop o notebook configurato con i componenti della UI;
- Tramite connessione remota ad una UI di infrastruttura o di gruppo.

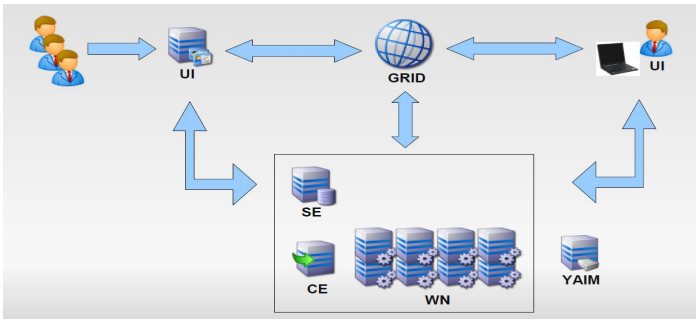



**Figura 18: Metodi di accesso ad un sito Grid gLite**

**[Rif.] Sito SCoPE: http://www.scope.unina.it/default.aspx**

[1]Job: rappresenta l'attività computazionale che si vuole svolgere sulla griglia.

Quindi, grazie ad una UI è possibile interagire con un sito Grid e sfruttarne le risorse. La modalità di base per interagire con una griglia, impiega le linee di comando testali cioè le CLI (Command Line Interface), implementate per sistemi Unix-like. Ad esempio, il seguente comando qui di seguito presentato, permette di creare un certificato proxy:

*[gvebber@notredame ~]$ voms-proxy-init --voms unina.it*
*Enter GRID pass phrase:*
*Your identity: /C=IT/O=INFN/OU=Personal Certificate/L=Federico II/CN=Giovanni Vebber*
*Creating temporary proxy ..... Done*
*Contacting voms01.scope.unina.it:15003 [/C=IT/O=INFN/OU=Host/L=Federico II/CN=voms01.scope.unina.it] "unina.it" Done*
*Creating proxy ..... Done*
*Your proxy is valid until Thu Jan 13 00:23:57 2011*

Anche dall'esempio si comprende che le interfaccia CLI non sono molto user-friendly, anzi il dover ricordare, i comandi, la loro sequenza, le eventuali opzioni, è causa di facili e frequenti errori (in pratica non sono di facile utilizzo per utenti poco esperti). Si vedrà nel capitolo 5 come si è automatizzato l'uso di tali comandi grazie all'ausilio delle system call realizzate in ambiente Java, che garantiscono una totale trasparenza per gli utenti che vogliono svolgere attività sulla griglia.

## 2.5. Il progetto SCoPE-Grid

Il progetto SCoPE[1], nasce come iniziativa dell'Università Federico II di Napoli, con l'obiettivo di creare una infrastruttura di supercalcolatori, fondata sul concetto di Grid computing, utile soprattutto per il supporto verso attività di ricerca, ma non solo.
L'architettura di SCoPE (che si basa sul progetto INFNGRID[2]), è stata sviluppata con lo scopo primario di integrare le attività di calcolo e archiviazione delle maggiori sedi di ricerca della Federico II, inglobandole in un unica piattaforma Grid, che poi a sua volta, va ad integrarsi con altre griglie nazionali e internazionali. Ad oggi molte attività di ricerca sono sviluppate attraverso SCoPE, attività che vanno dal settore astrofisico, matematico, informatico, statistico, sociale, e tanti altri.
La griglia SCoPE (che si basa sul middleware gLite 3.2), fornisce all'utente una serie di risorse che possono essere sfruttate per eseguire sia attività di calcolo/elaborazione che di archiviazione. Ovviamente le varie risorse sono vincolate alle virtual organization a cui l'utente che vuole sfruttarle deve appartenere. L'elenco qui presentato mostra le risorse di Computing Element e Storage Element, disponibili nella griglia SCoPE.

**Risorse di Computing Element**:

*cecream-cyb.ca.infn.it:8443/cream-lsf-unina*
*ce01.scope.unina.it:8443/cream-pbs-uninacert*
*ce01.scope.unina.it:8443/cream-pbs-unina_hpc*
*ce02.scope.unina.it:8443/cream-pbs-unina_hpc*
*ce02.scope.unina.it:8443/cream-pbs-uninacert*
*ce01.scope.unina.it:8443/cream-pbs-unina_long*
*ce02.scope.unina.it:8443/cream-pbs-unina_long*
*ce02.scope.unina.it:8443/cream-pbs-unina_short*
*ce01.scope.unina.it:8443/cream-pbs-unina_short*
*ce-cyb.ca.infn.it:2119/jobmanager-lcglsf-unina*
*ce01.scope.unina.it:8443/cream-pbs-unina_infinite*
*ce02.scope.unina.it:8443/cream-pbs-unina_infinite*
*chimce01.campusgrid.unina.it:2119/jobmanager-lcgpbs-scopecert*
*chimce01.campusgrid.unina.it:2119/jobmanager-lcgpbs-scope_long*
*chimce01.campusgrid.unina.it:2119/jobmanager-lcgpbs-scope_short*
*chimce01.campusgrid.unina.it:2119/jobmanager-lcgpbs-scope_infinite*
*unime-ce-01.me.pi2s2.it:2119/jobmanager-lcglsf-unina_long*
*unime-ce-01.me.pi2s2.it:2119/jobmanager-lcglsf-unina_short*
*unime-ce-01.me.pi2s2.it:2119/jobmanager-lcglsf-unina_infinite*
*unict-dmi-ce-01.ct.pi2s2.it:2119/jobmanager-lcglsf-unina_long*
*unict-dmi-ce-01.ct.pi2s2.it:2119/jobmanager-lcglsf-unina_short*
*unict-dmi-ce-01.ct.pi2s2.it:2119/jobmanager-lcglsf-unina_infinite*
*grisuce.scope.unina.it:8443/cream-pbs-unina_long*

[1]SCoPE: Sistema Cooperativo distribuito ad alte Prestazioni per Elaborazioni scientifiche
[2]INFNGRID: Istituto Nazionale di Fisica Nucleare progetto Grid

*grisuce.scope.unina.it:8443/cream-pbs-unina_short*
*grisuce.scope.unina.it:8443/cream-pbs-unina_infinite*
*ce.scope.unina.it:8443/cream-pbs-egee_long*
*ce.scope.unina.it:8443/cream-pbs-egee_short*
*scopedma-ce.dma.unina.it:2119/jobmanager-lcgpbs-scopecert*
*scopedma-ce.dma.unina.it:2119/jobmanager-lcgpbs-scope_long*
*scopedma-ce.dma.unina.it:2119/jobmanager-lcgpbs-scope_short*
*scopedma-ce.dma.unina.it:2119/jobmanager-lcgpbs-scope_infinite*

**Risorse di Storage Element**:

*se01.scope.unina.it*
*se-cyb.ca.infn.it*
*se.scope.unina.it*
*se.scope.unina.it*
*inaf-se-01.ct.pi2s2.it*
*inaf-se-01.ct.pi2s2.it*
*grisuse.scope.unina.it*
*grisuse.scope.unina.it*
*unict-dmi-se-01.ct.pi2s2.it*
*unict-dmi-se-01.ct.pi2s2.it*
*storagesrv1.ceinge.unina.it*
*unime-se-01.me.pi2s2.it*
*unime-se-01.me.pi2s2.it*

# 3 Tecnologie software impiegate

In questo capitolo vengono presentate le tecnologie software impiegate per ottenere una User Interface da utilizzare per poter accedere alle risorse del sito SCoPE-Grid. Infatti prima di procedere con l'analisi della progettazione ed implementazione delle nuove funzionalità (modulo DriverGrid), per il componente DRMS, occorre installare e configurare una User Interface per accedere alla griglia SCoPE. Come già detto in precedenza, la UI è una componente del middleware gLite, utilizzata come strumento per ottenere un punto d'accesso verso una griglia. La distribuzione gLite presenta più versioni, che variano, sia per come sono implementate a livello software, sia per i dispositivi hardware che possono gestire. Esistono ad esempio versioni gLite per macchine con architettura a 32/64 bit e con versioni di sistemi operativi (di base UNIX), differenti. La versione gLite (nel caso presente INFNGRID-gLite), qui impegata è la 3.2 con architettura a 64 bit. Si è scelta tale versione di gLite, primo perché più stabile nel suo complesso, ma soprattutto pienamente supportata dalla griglia di SCoPE (che si basa sul progetto INFNGRID), a differenze delle altre versioni.

## 3.1. Installazione gLite middleware 3.2 (Nodo UI)

Per installare gLite 3.2 (nodo UI) su una macchina (pc, notebook), occorre prima di tutto disporre di un sistema operativo Scientific Linux 5.5[1]. Inoltre, bisogna verificare che nel sistema operativo installato, il protocollo NTP e il logrotate siano funzionanti, dato che risultano necessari per i componenti da installare. L'NTP serve per permettere ai componenti (chiamati anche nodi di gLite), di essere sincronizzati (se i nodi sono di tipo AFS sono già sincronizzati). Logrotate invece serve per gestire file di log per un'applicazione. La fase di installazione vera e propria di gLite (nodo UI), avviene attraverso il gestore dei pacchetti (yum). Yum è un programma che permette di installare, aggiungere, aggiornare e cancellare pacchetti rpm[2]. In pratica yum è in grado di reperire programmi/librerie, di cui necessita il pacchetto, che si ha intenzione di installare. Per una corretta installazione di gLite (nodo UI), è necessario configurare il gestore dei pacchetti (yum), con i seguenti repositories:

- ***The middleware repositories***: identifica il componente gLite che si vuole installare. In pratica rappresenta il file che permette di andare a reperire le informazioni per il componente gLite, in questo caso le informazioni sul nodo riferito alla User Interface.
- ***the CA repository***: Tale repository serve per mantenere aggiornata la lista dei Certification Authority (CA), necessaria per il nodo installato. Infatti sia l'elenco che la struttura della CA può variare in modo indipendente dal middleware sottostante.
- ***DAG***: Il DAG è un repository che mantiene una serie di pacchetti che non sempre sono resi disponibili attraverso le versioni di Scientific Linux.
- ***INFNGRID gLite3.2(x86_64) repository***: Infine come ultimo file, va aggiunto al gestore yum, il repository da dove si reperiscono i pacchetti per l'installazione del software gLite.

In sequenza, viene di seguito presentato, il contenuto dei file repository, precedentemente descritti:

**The middleware repositories:** ***"glite-ui.repo"***

```
#
# gLite UI repositories
#
[glite-UI_sl5_x86_64_release]
name = gLite UI 3.2 x86_64 (release)
baseurl = http://grid-it.cnaf.infn.it/mrepo/glite_sl5-x86_64/RPMS.ui-release/
enabled = 1
protect = 0
[glite-UI_sl5_x86_64_updates]
name = gLite UI 3.2 x86_64 (updates)
baseurl = http://grid-it.cnaf.infn.it/mrepo/glite_sl5-x86_64/RPMS.ui-updates/
```

[1]Sito web Scientific Linux: https://www.scientificlinux.org/
[2]RPM: Red Hat Package Manager

*enabled = 1*
*protect = 0*
*[glite-UI_sl5_x86_64_externals]*
*name = gLite UI 3.2 x86_64 (externals)*
*baseurl = http://grid-it.cnaf.infn.it/mrepo/glite_sl5-x86_64/RPMS.ui-externals/*
*enabled = 1*
*protect = 0*

**the CA repository:** ***"lcg-ca.repo"***

*#*
*# CA repository*
*#*
*[CAs]*
*name = CAs*
*baseurl = http://linuxsoft.cern.ch/LCG-CAs/current*
*enabled = 1*
*protect = 0*

***DAG*: "*dag.repo"***

*#*
*# DAG (http://dag.wieers.com) additional RPMS repository*
*#*
*# CERN does NOT provide support for packages in this repository.*
*#*
*[dag]*
*name = DAG (http://dag.wieers.com) additional RPMS repository*
*baseurl = http://linuxsoft.cern.ch/dag/redhat/el5/en/$basearch/dag*
*gpgkey = http://linuxsoft.cern.ch/cern/slc5X/$basearch/RPM-GPG-KEYs/RPM-GPGKEY-dag*
*gpgcheck = 1*
*enabled = 1*
*protect = 0*

**INFNGRID gLite3.2(x86_64) Repository: "*ig.repo"***

*#*
*# INFNGRID repositories*
*#*
*[ig_sl5_x86_64]*
*name = INFNGRID 3.2 x86_64*
*baseurl = http://grid-it.cnaf.infn.it/mrepo/ig_sl5-x86_64/RPMS.3_2_0/*
*enabled = 1*
*protect = 0*
*[ig_sl5_x86_64_externals]*
*name = INFNGRID 3.2 x86_64 (externals)*
*baseurl = http://grid-it.cnaf.infn.it/mrepo/ig_sl5-x86_64/RPMS.3_2_0_externals/*
*enabled = 1*
*protect = 0*

Tutti questi file repository vanno aggiunti nella directory utilizzata dal gestore(yum). Di norma tale directory è ***/etc/yum.repos.d***. Una volta terminata la procedura di aggiornamento per il gestore yum, ottenuta attraverso l'istruzione "**yum update**". E' finalmente possibile installare in successione con i seguenti comandi, "**yum install lcg-CA**" e "**yum groupinstall ig_UI_noafs**", sia la lista Certification Authority aggiornata, che il nodo relativo alla User Interface.

## 3.2. Configurazione gLite middleware 3.2 (Nodo UI)

La configurazione di un nodo gLite avviene attraverso le funzionalità di YAIM[1] (nel caso presente INFNGRID-YAIM). L'obiettivo di YAIM è quello di fornire un sistema di configurazione semplice per i componenti (nodi), del middleweare gLite. YAIM è un insieme di script bash[2] e funzioni, ed è distribuito attraverso pacchetti rpm, residente solitamente nella directory ***/opt/glite/yaim***. Configurare un nodo significa modificare uno o più file (secondo la sintassi bash), con i parametri relativi al sito Grid a cui accedere. Una volta modificati tali file, essi vengono eseguiti attraverso i moduli YAIM. Vediamo adesso come configurare un nodo e nello specifico la UI riferita al sito SCoPE-Grid.
Per prima cosa occorre creare una directory <*confdir*> (di solito in ***/opt/glite/yaim***), dove inserire i file che serviranno per la configurazione del nodo. Con l'installazione di YAIM, per evitare di dover creare ex-novo i file, ne vengono presentati alcuni già configurati (disponibili in ***opt/glite/yaim/examples/siteinfo/***), dove occorre solo che si inseriscano i parametri della griglia a cui si vuole accedere. Tali parametri sono forniti dall'amministratore di sistema dell'infrastruttura Grid. Di seguito vengono presentati sinteticamente i file utili per accedere correttamente alla griglia di SCoPE.

### 3.2.1 Descrizione file <your-site-info.def>

Rappresenta il principale file di configurazione, comprende una lista di variabili (coppia chiave-valore), utili per stabilire la connessione all'infrastruttura a cui agganciarsi. Vediamo una lista di alcune delle variabili modificate per accedere al sito SCoPE-Grid.

**Variabili generali di configurazione:**

Rappresentano le variabili che identificano il sito Grid, ad esempio la lista dei Worker Node, della Virtual Organization, oppure il parametro del percorso (path), dove viene inserito il file che la griglia restituisce in output, dopo aver terminato un'elaborazione.

```
WN_LIST=/opt/glite/yaim/config/wn-list.conf
USERS_CONF=/opt/glite/yaim/config/ig-users_EGEE_TORQUE_WN.conf
GROUPS_CONF=/opt/glite/yaim/config/ig-groups_EGEE_TORQUE_WN.conf
FUNCTIONS_DIR=/opt/glite/yaim/functions

GSSKLOG=no
GSSKLOG_SERVER=my-gssklog.$MY_DOMAIN

OUTPUT_STORAGE=/tmp/jobOutput

# Site-wide settings
SITE_EMAIL=grid-prod@mlserver.unina.it
SITE_CRON_EMAIL=$SITE_EMAIL
SITE_SUPPORT_EMAIL=$SITE_EMAIL
SITE_NAME=UNINA-SCOPE-DATACENTER
SITE_LOC="Naples, Italy"
SITE_LAT=40.83785# # -90 to 90 degrees
SITE_LONG=14.18287# # -180 to 180 degrees
SITE_WEB="http://www.scope.unina.it"
SITE_OTHER_GRID="EGEE"
SITE_OTHER_EGEE_ROC="Italy"
```

**Variabili di configurazione Myproxy:**

Sono le variabili utilizzate per creare i certificati proxy con cui gli utenti si autenticano alla griglia.

```
PX_HOST=myproxy01.$MY_DOMAIN

GRID_TRUSTED_BROKERS="
'/C=IT/O=INFN/OU=Host/L=Federico II/CN=wms01.scope.unina.it'"
```

[1]Yaim: Yet Another Installation Manager
[2]Bash: shell testuale usata nei sistemi Unix e Unix-like

**Variabili per la gestione del Computing Element:**

Variabili usate principalmente per accedere alle risorse computazionali.

*DGAS_HLR_RESOURCE="hlr01.$MY_DOMAIN"*
*DGAS_JOBS_TO_PROCESS="grid"*
*DGAS_IGNORE_JOBS_LOGGED_BEFORE="2009-05-01"*
*DGAS_USE_CE_HOSTNAME="ce02.scope.unina.it"*

**Variabili configurazione Storage Element:**

Definiscono i parametri per gestire lo Storage Element della griglia.

*CLOSE_SE_HOST=se01.scope.unina.it*
*CLOSE_SE_HOST_EGEE=se.scope.unina.it*

*SE_LIST="$DPM_HOST"*
*SE_ARCH="multidisk" # "disk, tape, multidisk, other"*

Riferito alla griglia SCoPE tale file è così rinominato ***WN-scope-site-info_EGEE.def***.

### 3.2.2 Descrizione file <your-wn-list.conf>

Rappresenta la lista dei Worker Node nel formato “hostname.domainname” indicate per righe. Tale file è definito nella variabile WN_LIST presente in <*your-site-info.def*> e viene fornito dall'amministratore del sito Grid. Ecco un esempio di come si presenta tale file per la griglia SCoPE:

*wn001.scope.unina.it*

### 3.2.3 Descrizione file <your-users.conf>

Definisce la mappatura degli utenti ed è modificato in base alle politiche adottate da ogni sito Grid . Viene fornito dall'amministratore ed è definito nella variabile *USERS_CONF* del file <*your-site-info.def*>. Ecco un esempio di come si presenta per la griglia SCoPE:

*10001:unina.it001:10000:unina.it:unina.it::*

### 3.2.4 Descrizione file <your-groups.conf>

Identifica la mappatura della VOMS[1], e in definitiva dei gruppi delle varie Virtual Organization. E' presente nella variabile *GROUPS_CONF* del file <*your-site-info.def*> e fornito dall'amministratore. Esempio riferito al sito SCoPE:

```
"/VO=cometa/GROUP=/cometa/ROLE=lcgadmin":::sgm:
"/VO=cometa/GROUP=/cometa/ROLE=production":::prd:
"/VO=cometa/GROUP=/cometa"::::
"/VO=cresco/GROUP=/cresco/ROLE=lcgadmin":::sgm:
"/VO=cresco/GROUP=/cresco/ROLE=production":::prd:
"/VO=cresco/GROUP=/cresco"::::
"/VO=cybersar/GROUP=/cybersar/ROLE=lcgadmin":::sgm:
"/VO=cybersar/GROUP=/cybersar/ROLE=production":::prd:
"/VO=cybersar/GROUP=/cybersar"::::
"/VO=poncert/GROUP=/poncert"::::
"/VO=matisse/GROUP=/matisse/ROLE=lcgadmin":::sgm:
"/VO=matisse/GROUP=/matisse/ROLE=production":::prd:
"/VO=matisse/GROUP=/matisse"::::
"/VO=unina.it/GROUP=/unina.it/ROLE=lcgadmin":::sgm:
"/VO=unina.it/GROUP=/unina.it/ROLE=production":::prd:
"/VO=unina.it/GROUP=/unina.it"::::
"/VO=spaci/GROUP=/spaci/ROLE=lcgadmin":::sgm:
"/VO=spaci/GROUP=/spaci/ROLE=production":::prd:
"/VO=spaci/GROUP=/spaci"::::
"/VO=atlas/GROUP=/atlas/ROLE=lcgadmin":::sgm:
```

[1]VOMS: Virtual Organization Membership Service

```
"/VO=atlas/GROUP=/atlas/ROLE=production":::prd:
"/VO=atlas/GROUP=/atlas"::::
```

### 3.2.5 Configurazione User Interface

Dopo aver preparato i relativi file, per configurare un nodo e nel caso specifico il nodo UI, basta eseguire tale comando:

- ***/opt/glite/yaim/bin/ig_yaim -c -s <your-site-info.def> -n <nodetype>***

In pratica il modulo *ig_yaim* prende dal file <*your-site-info.def*> i parametri e li utilizza per configurare il nodo designato (in questo caso la UI).

Per il sito SCoPE-Grid, <*your-site-info.def*> si sostituisce con ***WN-scope-site-info_EGEE.def*** e <nodetype> con ***ig_UI_noafs***.

# 4 Scelte progettuali definite

Nei capitoli precedenti si è analizzato e prima spiegato, come funziona una griglia, identificando le sue caratteristiche sia software che hardware. In seguito, si è mostrato come installare e configurare una User Interface, per poter accedere alle risorse del sito SCoPE-GRID. Invece, nel presente capitolo attraverso le metodologie dell'ingegneria del software (requisiti funzionali, use case, tabelle di cockburn, etc), si andrà ad individuare tutta la logica di funzionamento del progetto (applicazione), DAMEGRID.

## 4.1. Analisi del progetto

Il progetto (denominato DAMEGRID), prevede la realizzazione o meglio il restyling del componente driver (DRMS[1]), della suite DAME, integrando tale componente con delle nuove funzionalità (modulo DriverGRID), che permettano di effettuare operazioni sulla infrastruttura SCoPE-GRID.
L'idea di base del progetto è stata quella di suddividere il driver (per l'attività che permette di eseguire e cancellare esperimenti), in due livelli. Il primo livello è rivolto alle funzionalità che servono a individuare il sottosistema (GRID-SA), da utilizzare. Il secondo livello invece, è lo strato che identifica le funzionalità per il sottosistema GRID e quelle per il sottosistema SA. Per un quadro più chiaro si veda la figura 19:

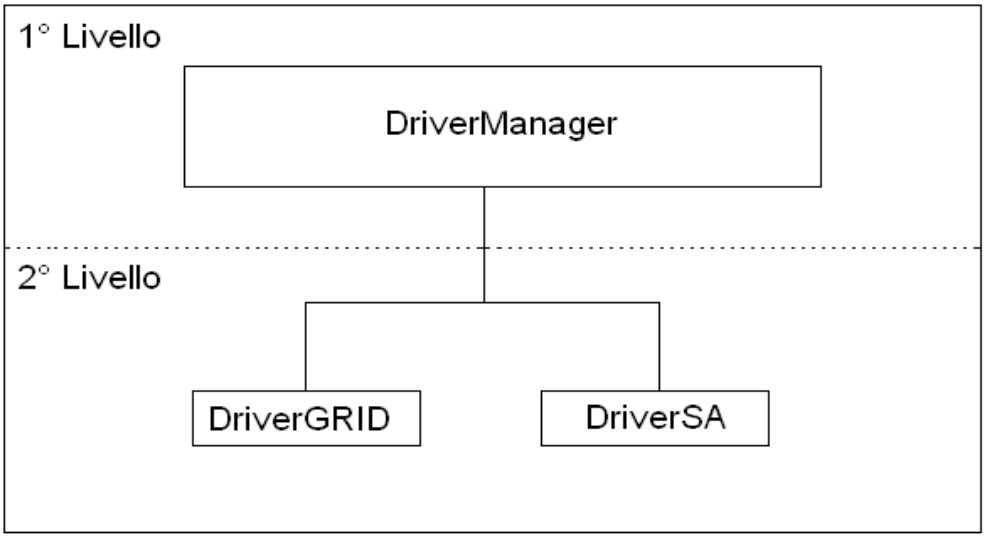



**Figura 19: Stratificazione componente DRMS**

Il DriverManager è il subcomponente che al primo livello seleziona e invoca la piattaforma da utilizzare.
In realtà la funzionalità di selezione della piattaforma viene attivata se la richiesta, che arriva dal componente Framework della suite DAME, è quella di lanciare un esperimento. Mentre se ad esempio viene richiesta la cancellazione di un job presente sulla griglia, il DriverManager evita l'attività di selezione della piattaforma e invoca direttamente il modulo del secondo livello (in questo caso il DriverGRID). L'attività di selezione del sottosistema, avviene attraverso uno scheduler, che analizzando una serie di parametri, determina quale piattaforma utilizzare. In questa fase del progetto si prevede solo l'impostazione dello scheduler senza la sua reale implementazione.

Il DriverGRID è il subcomponente che al secondo livello gestisce le funzionalità, per eseguire le varie operazioni sulla griglia. Le funzionalità sono:

- Invio di job sulla griglia;
- Cancellazione di un job dalla griglia;
- Creazione e gestione certificato proxy e myproxy;
- Output di un job;
- Stato di un job.

[1]DRMS: Driver Management System

Infine, l'altro subcomponente presente al secondo livello è il DriverSA che svolge le attività richieste sulla macchina stand alone (già implementato e funzionante nella versione beta di DAMEWARE).

## 4.2. Analisi dei requisiti

Attraverso l'analisi del progetto, vengono adesso elencati i requisiti funzionali, cioè l'elenco dei servizi, o funzioni, offerti dal sistema.

### 4.2.1 Requisiti funzionali

- Il sistema deve selezionare, attraverso l'analisi di determinati parametri (scheduler), la piattaforma da utilizzare;
- Il sistema deve chiamare il componente DriverGRID;
- Il sistema deve chiamare il componente DriverSA;
- Il sistema deve creare un certificato proxy e il myproxy;
- Il sistema deve creare un file in formato jdl (con l'esperimento da eseguire), e inoltrare tale file alla griglia;
- Il sistema deve ritornare lo stato del job. Tale stato può essere Submitted, Waiting, Ready, Scheduled, Running, Done, Aborted, Cancelled, Cleared;
- Il sistema deve cancellare il job richiesto dalla griglia;
- Il sistema deve, una volta terminato con successo un job, recuperare i file di l'output, direttamente o attraverso lo Storage Element della griglia, e renderli disponibili all'utente che ha lanciato un esperimento.

## 4.3. Descrizione file JDL

L'invio di attività (nel caso specifico gli esperimenti richiesti), sulla griglia avviene attraverso un file (con estensione jdl), usato per descrivere i job e aggregare le informazioni. Il Job Description Language è un linguaggio di alto livello, usato dal middleware gLite per descrivere i requisiti sui vari job che dovranno essere elaborati dalle risorse della griglia. Un file JDL si basa su una serie di linee che esprimono nel formato "*attribute = expression*", i requisiti dei job che dovranno essere elaborati. Nella tabella qui presentata, vengono elencati i principali attributi di cui può essere composto un file JDL:

| Attributi | Obbligatorio | Attività | Esempio |
|---|---|---|---|
| Executable | SI | Specifica l'eseguibile da processare | Executable = "test.sh"; |
| Arguments | NO | Elenco dei possibili argomenti utli per l'eseguibile | Arguments = "hello 10"; |
| StdOutput | SI | Specifica lo standard output | StdOutput = "std.out"; |
| StdError | SI | Specifica lo standard error | StdError = "std.err"; |
| StdInput | NO | Specifica lo standard input | StdInput = "std.in"; |
| InputSandbox | NO | Trasferisce i file di Input dalla UI al Worker Node | InputSandbox = {"test.sh","std.in"}; |
| OutputSandbox | NO | Trasferisce i file di Output dal WN alla UI | OutputSandbox = {"std.out","std.err"}; |
| Environment | NO | Estensione dell'ambiente di esecuzione del job | Environment = {"CMS_PATH=$HOME/cms","CMS_DB=$CMS_PATH/cmdb"}; |
| Requirements | NO | Esprime dei costraints sulle risorse su cui il job viene eseguito | Requirements = other.GlueCEInfoLRMSType == "PBS"; |
| OutputData | NO | Copia il file di Output dal WN allo Storage Element | OutputData = {<br>[<br>OutputFile="mc.out";<br>LogicalFileName="lfn:/grid/unina.it/Dame/mc.out";<br>StorageElement = "se01.scope.unina.it";<br>]<br>}; |
| InputData | NO | Legge un file dallo Storage Element, utile per l'eseguibile | InputData = {LogicalFileName="lfn:/grid/unina.it/Dame/input.txt"}; |

**Tabella #1: Descizione file JDL**

I file JDL presentano strutture sintattiche molto rigide, quindi occorre prestare molta attenzione a come vengono inseriti i vari attributi, onde evitare facili errori di sintassi.

## 4.4. Diagramma dei Casi d'uso

Vengono ora presentati attraverso i diagrammi use case e le tabelle di cockburn (per le tabelle consultare l'appendice A), le funzionalità che devono essere offerte dal componente Driver (DRMS) modificato, soprattutto per quanto riguarda quelle indicative del subcomponente DriverManager e del DriverGRID.

### 4.4.1 Use Case Diagram

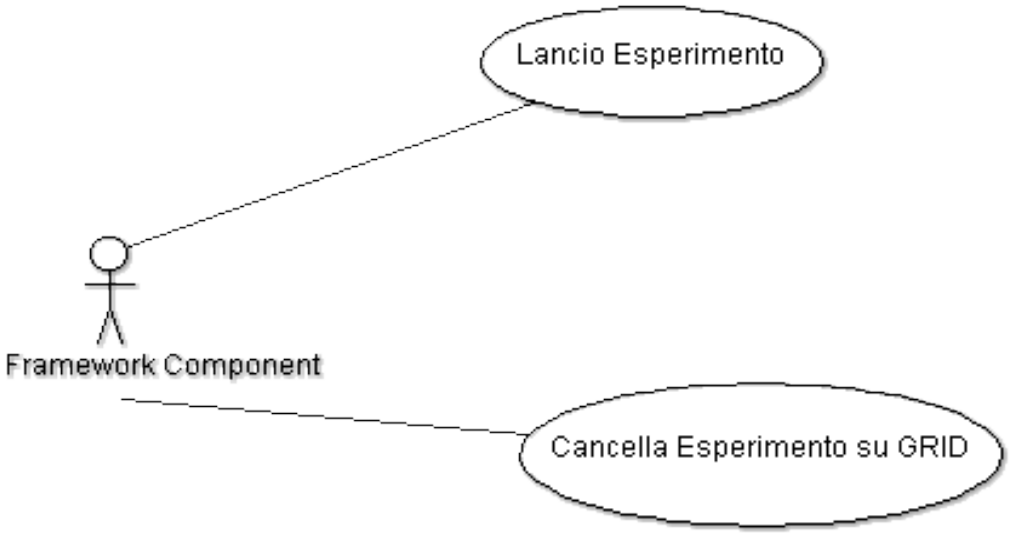



**Figura 20: Use Case Attività Framework**

Il diagramma dei casi d'uso sopra raffigurato rappresenta l'attività che il componente ***Framework*** deve effettuare affinché si possa interagire con il componente DRMS, permettendo in primis l'interazione con il modulo DriverManager e DriverGRID.

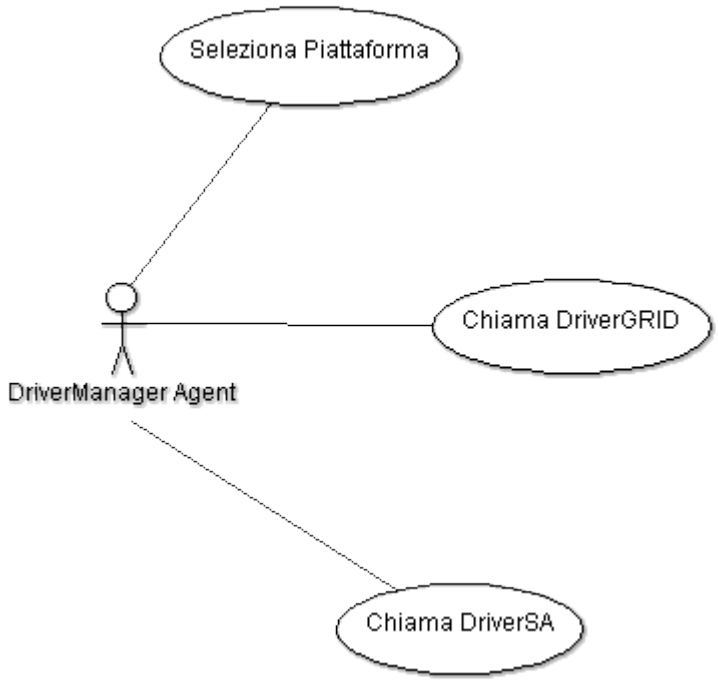



**Figura 21: Use Case DriverManager**

Il caso d'uso ***DriverManager***, come previsto in fase di analisi, prevede due macro attività. La prima rivolta alla selezione della piattaforma da utilizzare, che si ottiene attraverso un meccanismo di scheduling, valutando dei parametri forniti attraverso un file di configurazione esterno.
La seconda attività invece, permette di invocare (chiamare), il modulo DriverGRID oppure il DriverSA, in base alla piattaforma selezionata. Tale chiamata può avvenire anche direttamente (cioè evitando la selezione della piattaforma), se, ad esempio, la richiesta arrivata dal Framework fosse quella della cancellazione di un job dalla griglia.

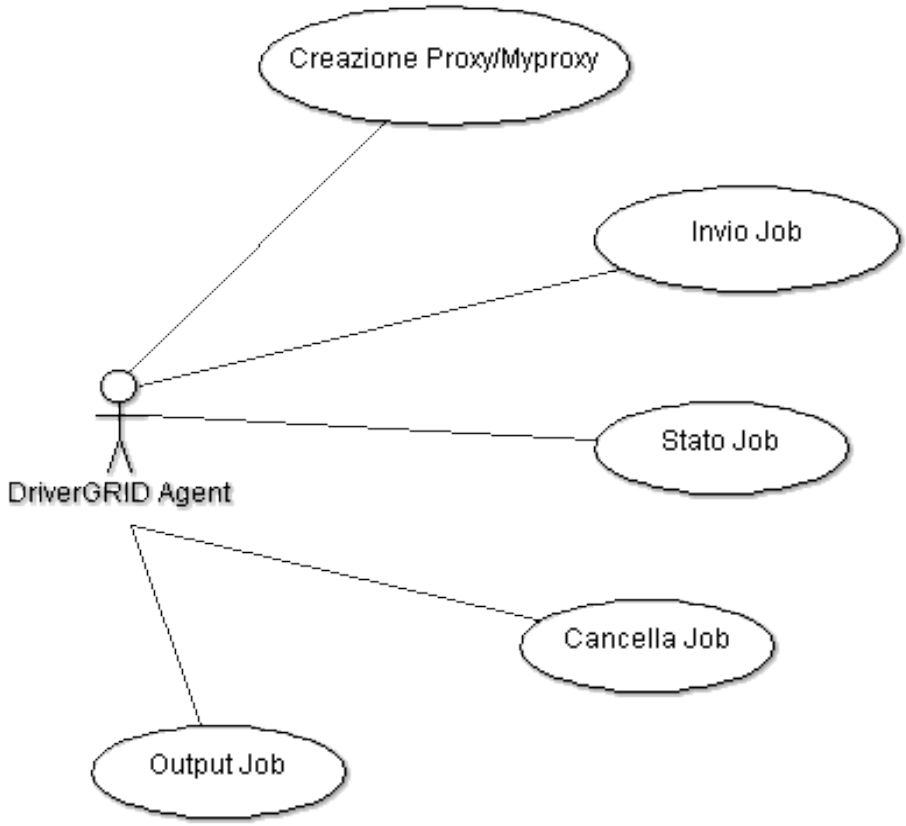



**Figura 22: Use Case DriverGRID**

Il caso d'uso ***DriverGRID*** identifica le funzionalità che devono essere utilizzate per svolgere attività sulla griglia. Come si può vedere, nel caso d'uso sono presenti attività come la creazione di un proxy, che serve per avere l'accesso alla griglia per circa 12 ore. Oltre al proxy viene creato un myproxy che permette di allungare la durata dell'accesso alla griglia per circa 7 giorni, fornendo così maggiore possibilità che determinati job non decadano anticipatamente causa scadenza del proxy. Le altre funzionalità del caso d'uso rappresentano le restanti attività che possono essere eseguite sulla griglia.

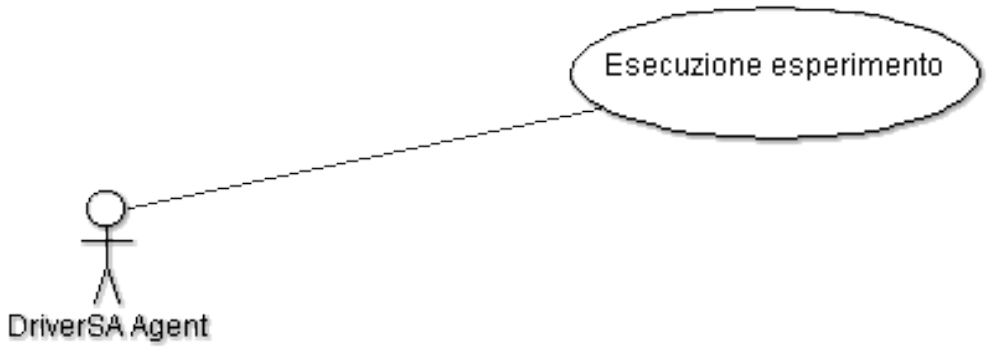



**Figure 23: Use Case DriverSA**

Il caso d'uso ***DriverSA*** rappresenta l'attività di esecuzione di un esperimento su macchina stand alone, dove più esperimenti vengono eseguiti in multithreading.

# 5 Design e specifiche tecniche

Nel capitolo precedente si sono analizzate le caratteristiche che l'applicazione deve integrare, cioè si sono individuati i requisiti funzionali. Nel presente capitolo si andranno ad individuare le specifiche architetturali e tecniche, attraverso l'individuazione delle classi e del flusso di comunicazione tra le varie componenti della suite DAME. Verranno anche descritte le specifiche tecniche adottate, per utilizzare le funzionalità della User Interface (comandi gLite), e per la gestione degli errori. Inoltre, viene presentata una breve descrizione, dell'attività di testing e delle altre funzionalità che il componente DRMS gestisce, oltre a quella di eseguire e cancellare esperimenti.

## 5.1. Descrizione modifiche al componente DRMS

Come già discusso in parte, nel capitolo precedente, l'obiettivo del progetto DAMEGRID prevede la realizzazione o meglio il restyling del componente driver (DRMS[1]), della suite DAME, integrando tale componente con delle nuove funzionalità (modulo DriverGrid), che permettono di effettuare operazioni sulla infrastruttura SCoPE-GRID. Per comprendere meglio come il DRMS gestisca le nuove attività, viene presentato uno schema che mostra le nuove funzionalità per il componente DRMS, individuando anche il flusso di comunicazione con le altre componenti della suite DAME.

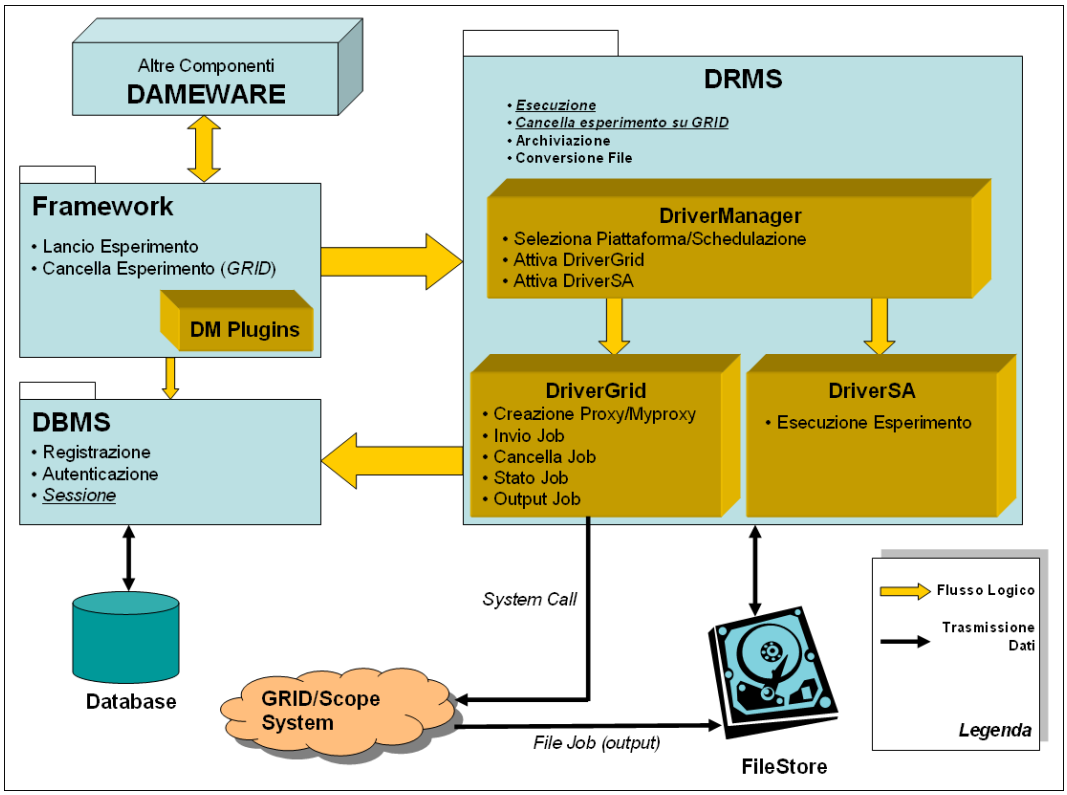



**Figura 24: Componente DRMS modificato**

Il DRMS (per l'attività che permette di eseguire e cancellare esperimenti), comunica con il componente ***Framework***, da cui riceve le richieste delle attività da svolgere, che sono:

- Lancio di un esperimento;
- Cancellazione esperimento su GRID.

Con la richiesta di lancio di un esperimento da parte del ***Framework***, il DriverManager, per prima cosa seleziona la piattaforma da utilizzare attraverso un meccanismo di scheduling. Nel seguito si ipotizza che la selezione della piattaforma abbia dato come risultato l'uso della struttura GRID. Il DriverManager allora attiva il DriverGRID, che gestisce le varie funzionalità per comunicare con la griglia. Questa comunicazione con la griglia avviene attraverso una serie di system call, che permettono di interagire in modo trasparente con il sito SCoPE-GRID. Invece, nel caso del DriverSA (Driver Stand Alone su singola macchina), tale componente gestisce solo l'esecuzione di un esperimento

[1]DRMS: Driver Management System

sulla macchina stand alone, attraverso il DMPlugin. Nel caso che la richiesta da parte del ***Framework*** sia stata quella di cancellare un determinato esperimento sulla griglia, il DriverManager evita l'attività di scheduling e invoca direttamente il DriverGRID che utilizza la funzionalità che permette di cancellare un esperimento (job), dalla griglia. Infine ogni risultato di un esperimento, indipendentemente dal fatto che sia stato gestito sulla griglia oppure su stand alone, deve essere reso disponibile all'utente che ha lanciato l'esperimento[1].
Inoltre essendo l'applicazione fruibile in un ambiente multithreading, alcune delle funzionalità (come si vedrà nel class diagram), vengono gestite in modo sincronizzato evitando così possibili collisioni sulle informazioni che si vanno ad analizzare.
Da quanto analizzato, viene ora presentata l'analisi dell'architettura impiegata per sviluppare le funzionalità richieste.

## 5.2. Analisi dell'architettura

L'applicazione implementata, è stata realizzata secondo un'architettura che prevede delle classi centralizzate (gestori), che in base alle richieste provenienti dal componente ***Framework***, delegano l'elaborazione a determinate classi Java che sviluppano le funzionalità preposte.
L'idea di delegare più classi specifiche per svolgere una determinata funzionalità rispecchia il concetto di design di basso accoppiamento ed alta coesione. Per accoppiamento si intende il metro di analisi per verificare quanto fortemente una classe sia relata ad altre classi, cioè di quanto sia forte la dipendenza tra le classi. Dire che una classe è fortemente accoppiata si intende che è impossibile modificarne una senza modificare le altre.
Ecco perché è consigliabile durante la fase di design definire classi con basso accoppiamento, in modo da permettere di:

- capire il codice di una classe senza leggere i dettagli delle altre;
- modificare una classe senza che le modifiche comportino problemi alle altre classi;
- riusare facilmente una classe senza dover importare eventualmente altre classi.

In sintesi un basso accoppiamento migliora la gestione del codice.

Invece, la coesione è una misura di quanto siano fortemente affini le responsabilità (servizi offerti), di una classe. Con alta coesione si intende che ciascuna unità è responsabile solo di un determinato compito. Un'alta coesione quindi è la condizione desiderabile che si vuole ottenere, in modo da:

- comprendere meglio i ruoli della classe;
- riutilizzare una classe;
- permettere una facile gestione di una classe;
- limitare i cambiamenti nel codice.

La coesione e l'accoppiamento sono entrambe indispensabili, per cercare di ottenere un codice che rispecchi determinati standard qualitativi. Da quanto appena detto, si comprende meglio il motivo del perché il sistema realizzato, preveda questa struttura separata in moduli, dove ogni modulo svolge una ben definita attività.
Le classi che svolgono il ruolo di gestore, hanno il compito di utilizzare determinate funzionalità, come ad esempio la gestione dello scheduling, oppure la gestione dell'integrazione/comunicazione con il sistema GRID. Oltre alle classi gestore esistono poi le classi, che, come detto, hanno il compito di elaborare le varie funzionalità richieste. Ad esempio esistono le classi che permettono l'interazione, attraverso le system call, con l'infrastruttura SCoPE-GRID. Questa interazione con la griglia, deve permettere di creare un proxy, myproxy, inoltrare e cancellare un job, ottenerne lo stato e l'output, quest'ultimo se terminato in modo corretto.

Viene adesso presentato il sistema delle classi realizzato (cioè quali sono e che attività svolgono):

[1]In questa prima release tutti i file di output vengono copiati in una directory presente sulla macchina dove risiede la User Interface.

- **DriverManagerManagement Class**: Rappresenta la classe che gestisce le funzionalità di selezione della piattaforma e di attivazione del gestore DriverGRIDManagement o gestore DriverSAManagement;
- **DriverGridManagement Class**: E' la classe che gestisce le funzionalità che servono per interagire con il sistema SCoPE-GRID;
- **DriverSAManagement Class**: E' la classe che gestisce l'attività di esecuzione di un esperimento su macchina stand alone;
- **Proxy Class**: Rappresenta la classe che crea e gestisce il proxy, permettendo così di poter comunicare con il sistema SCoPE-GRID;
- **MyProxy Class**: E' la classe incaricata di creare e gestire il myproxy;
- **SubmitJob Class**: Rappresenta la classe che è incaricata di inviare alla GRID i job (esperimenti), che devono essere eseguiti;
- **OutputJob Class**: E' la classe che permette di gestire l'output di un job;
- **CancelJob Class**: Rappresenta la classe che si incarica di cancellare un determinato job presente sulla griglia;
- **StatusJob Class**: E' la classe che presenta la funzionalità di analizzare lo stato di un job in esecuzione;
- **Scheduler Class**: Rappresenta la classe che gestisce l'attività di selezione della piattaforma da utilizzare;
- **ThreadExperiment Class**: E' la classe che permette di istanziare un nuovo thread per ogni job (esperimento), che viene inoltrato alla griglia. Tale thread controlla periodicamente lo status del job (ogni minuto), ed in caso di terminazione corretta, ne fornisce l'output;
- **Parser Class**: Serve a gestire attraverso delle espressioni regolari prefissate, l'identificazione di una determinata frase o parola, presente nella stringa fornita come parametro di input per i vari metodi che compongono la classe;
- **Log Class**: Compito di tale classe è quello di fornire all'amministratore del sistema informazioni che descrivono:

    - ogni job inoltrato alla griglia;
    - eventuale errore verificatosi durante l'elaborazione del thread che gestisce l'esperimento;
    - creazione del certificato Proxy;
    - creazione del certificato MyProxy.

## 5.3. Class Diagram & Sequence Diagram

Nella fase di analisi dell'architettura si sono presentate le nuove classi che si vanno ad aggiungere al componente DRMS della suite DAME. Vediamo adesso da cosa sono composte (metodi, attributi), attraverso il diagramma delle classi. Come si vedrà nel diagramma alcuni metodi prevedono la parola “synchronized”, questo perché devono essere gestiti in mutua esclusione. Cioè essendo l'applicazione basata su struttura concorrente, è stato necessario definire delle politiche di accesso controllato a certe risorse. Inoltre per vedere più dettagliatamente anche le relazioni che intercorrono tra le classi consultare i Sequence Diagram nell'appendice B.

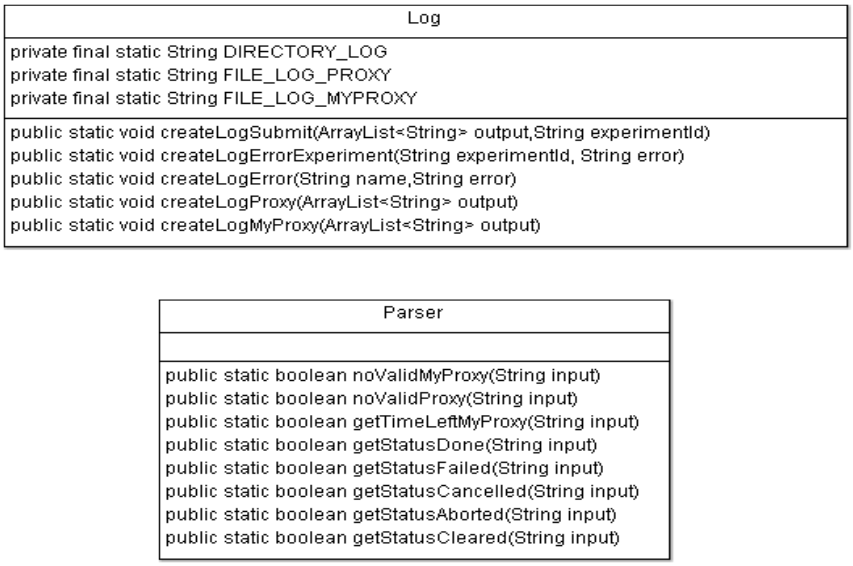



**Figura 25: Class Diagram 1**

**Figura 26: Class Diagram 2**

## 5.4. Analisi gestione status job

Durante l'arco di tempo in cui un job è presente sulla griglia, attraversa determinati stati che ne identificano l'attività. In pratica un job inoltrato alla griglia non viene immediatamente eseguito ma attraversa varie fasi, prima di entrare nello stato di *running*. E solo in tale condizione il job entra in esecuzione e vi rimane finché non è terminato con successo, o se si presentano degli errori (*aborted*), oppure se viene fatta richiesta di cancellazione del job da parte dell'utente. Indipendentemente dallo stato in cui si trova, alla fine della elaborazione il job assume sempre lo stato finale *cleared*. Nel seguente elenco e nel diagramma degli stati che segue, vengono presentati i vari stati di un job:

- **Submitted**: Il job è inoltrato dall'utente ma non ancora trasferito al Workerload Managment System;
- **Waiting**: Il job è accettato nella Task Queue (coda), e attende l'elaborazione da parte del Workerload Managment System;
- **Ready**: Il job viene eseguito dal Workerload Managment System ma non è ancora trasferito nel Computing Element (CE), scelto;
- **Scheduled**: Il job attende per l'esecuzione nella coda locale del CE;
- **Running**: il job è in fase di esecuzione;
- **Done**: l'attività del job è terminata o comunque si trova in uno stato terminale;
- **Cancelled**: Il job viene cancellato come richiesto dall'utente;
- **Cleared**: Il job entra nello stato cleared comunicando così al WMS che può eliminarlo dall'attività computazionale.

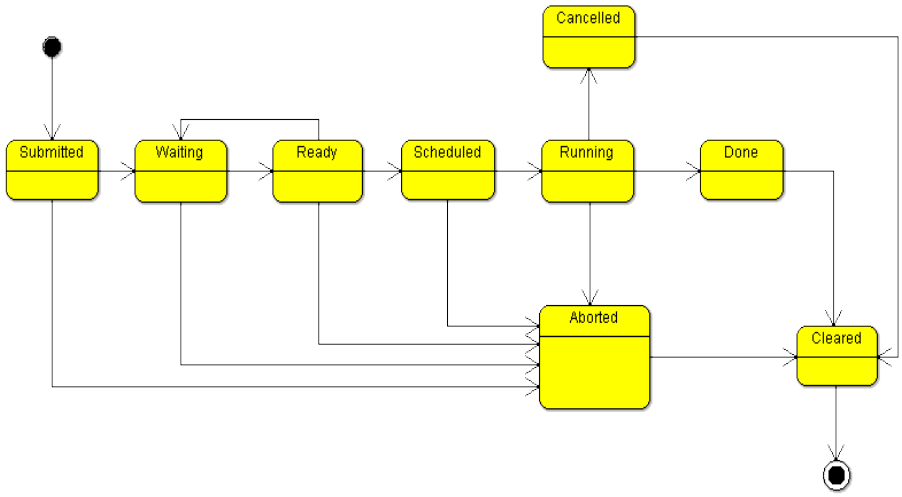



**Figura 27: Statechart Diagram**

### 5.4.1 Interrogazione status job

Si è visto che lo stato di un job varia durante l'elaborazione sulla griglia. Ovviamente di queste variazioni, l'utente che ha lanciato l'esperimento (sottoforma di job), deve essere informato. E' stato quindi necessario implementare una funzionalità che permettesse di interrogare periodicamente lo stato di un job. Tale attività viene svolta da un thread, che interroga ad intervalli di tempo definiti (1minuto), lo stato del job riportandone il valore che può essere presentato all'utente. Questo thread, lanciato dal subcomponente DriverGRID (classe DriverGridManagement), viene attivato per ogni job (esperimento), inviato alla griglia, e termina il suo percorso se si verifica una delle seguenti condizioni:

- lo stato del job ritorna il valore "**Done (Success)**";
- lo stato del job ritorna il valore "**Done (Failed)**";
- lo stato del job ritorna il valore "**Aborted**";
- lo stato del job ritorna il valore "**Cancelled**";
- lo stato del job ritorna il valore "**Cleared**".

Inoltre, prima che il thread termini la sua attività, se la condizione verificata è la prima elencata, allora viene anche gestita l'attività di output del job. Attività che rende disponibili, all'utente che ha lanciato l'esperimento, i file di output[1].

[1]In questa prima release tutti i file di output vengono copiati in una directory presente sulla macchina dove risiede la User Interface.

## 5.5. Comunicazione tra il DRMS e le altre componenti DAME

Vediamo come si presentano le comunicazioni aggiornate tra componenti della suite DAME da e verso il DRMS, e nello specifico per le attività di esecuzione e cancellazione degli esperimenti gestiti dal DriverGRID (classe DriverGridManagement).
Nei paragrafi precedenti si è visto che il componente ***Framework*** (che a sua volta riceve le richieste dal Front End che rappresenta l'ambiente di interazione con l'utente), comunica con il DRMS inviandogli le richieste di esecuzione di un esperimento oppure per la cancellazione di un determinato esperimento su GRID. A sua volta il DRMS (nello specifico il DriverGRID), con le nuove funzionalità, comunica attivamente con il ***DBMS***[1] (attraverso l'REDB[2]), effettuando interrogazioni al database per accedere oppure per inserire informazioni. Alcune delle comunicazioni principali che vengono svolte sono:

- Comunicazione tra il DRMS (DriverGRID), e ***DBMS*** per inserire nel campo della tupla, riferita all'esperimento in corso, il valore dell'identificativo del job. In pratica il percorso per risalire al job che si è inoltrato alla griglia;
- Comunicazione tra il DRMS (DriverGRID), e ***DBMS*** per modificare il campo status che descrive lo stato del job che si sta monitorando. In realtà, la reale comunicazione avviene tra il thread (classe ThreadExperiment), generato dal DriverGRID (vedi paragrafo 5.4.1), e il DBMS;
- Comunicazione tra il DRMS (DriverGRID), e ***DBMS*** per reperire l'identificativo del job che si vuole cancellare;
- Comunicazione tra il DRMS (DriverGRID), e ***DBMS*** per reperire le informazioni per la creazione del file jdl;
- Comunicazione tra il DRMS (DriverGRID), e ***DBMS*** per registrare le informazioni sui file di output, informazioni utili per l'utente che ha lanciato l'esperimento. Anche in questo caso la reale comunicazione avviene tra il thread (classe ThreadExperiment), generato dal DriverGRID (vedi paragrafo 5.4.1), e il DBMS.

Si comprende quindi che il DRMS modificato, per quanto riguarda l'attività di gestione di esperimenti (cancellazione ed esecuzione), continua a svolgere un'attività passiva verso il ***Framework*** da cui riceve le richieste. Invece ricopre un ruolo più attivo verso il ***DBMS*** a cui si interfaccia per chiedere o inviare informazioni. La figure 28 e 29 mostrano i flussi di comunicazione/informazione sia per il caso SA che per il caso GRID.

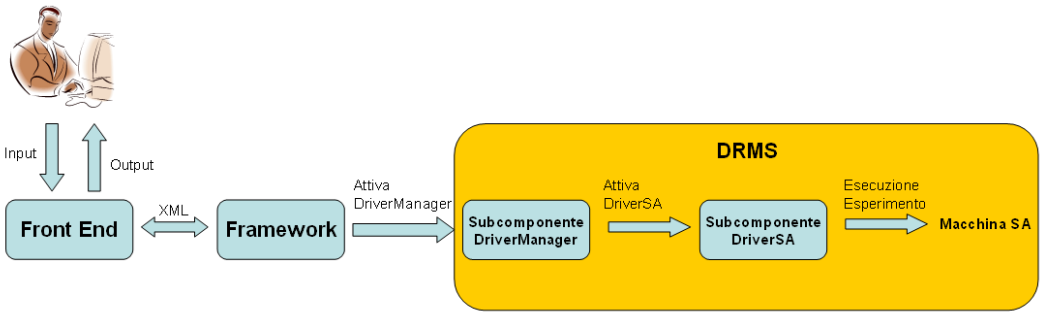



**Figura 28: Flusso informazioni DRMS su macchina SA**

Come si può vedere nel caso SA, il ***Framework*** chiama il subcomponente DriverManager (classe DriverManagerManagement), che a sua volta attraverso l'attività di scheduler, chiama il subcomponente DriverSA (classe DriverSAManagement), permettendo così l'esecuzione dell'esperimento richiesto su macchina SA.

[1]DBMS: Database Management System
[2]REDB - Registery & Database: interfaccia di comunicazione al database della suite DAME

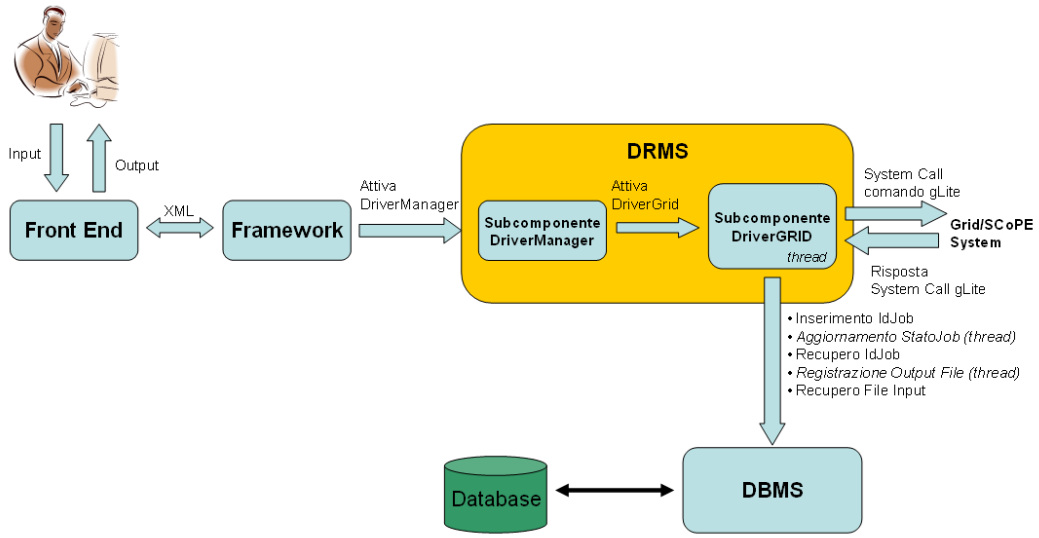



**Figura 29: Flusso informazioni DRMS in ambiente GRID**

Nel caso GRID invece, il ***Framework*** chiama il subcomponente DriverManager, che a sua volta attraverso l'attività di scheduler, chiama il subcomponente DriverGRID (classe DriverGridManagement). A questo punto vengono utilizzati i comandi di gLite per comunicare con la griglia e svolgere le attività richieste. Come detto in precedenza il subcomponente DriverGRID interagisce attivamente anche con il ***DBMS*** da cui richiede e invia informazioni.

## 5.6. Uso comandi gLite

Nei paragrafi precedenti si è visto come sono state realizzare le nuove funzionalità del DRMS. Non si è ancora discusso su come le attività sulla griglia vengono effettivamente eseguite, cioè come si utilizzano i vari comandi. Durante l'implementazione, tale fase è stata soggetto di molte versioni. In un primo momento si era pensato di utilizzare le funzionalità gLite attraverso delle API[1] Java. Il motivo principale di utilizzare delle API era nella maggiore facilità d'uso, dovuto all'utilizzo ad un più alto livello delle funzionalità di gLite, rispetto alle system call. In seguito, dopo una certo numero di prove si è visto che l'uso delle API provocava una serie di incompatibilità con il middleware gLite 3.2, problemi in parte dovuti, alla versione molto recente del middleware stesso. Inoltre non tutti comandi gLite erano gestibili tramite API, come ad esempio il comando che permette di valutare lo stato di un job. A quel punto, viste le incompatibilità e i limiti delle API, si è optato per implementare tutte le funzionalità utili di gLite, attraverso le system call. Ovviamente l'uso delle chiamate di sistema ha comportato una gestione integrale di tutte l'attività dello standard input, output ed error, cosa che invece con le API non era necessaria. La gestione attraverso le system call, offre però anche una serie di vantaggi, ad esempio è garantita una più rapida modifica di un eventuale comando, rispetto alle API. Cioè l'aggiornamento di un particolare comando (dovuto ad esempio ad una nuova versione di gLite), comporta solo l'eventuale modifica della system call che lo gestisce. Quindi, vero che le system call presentano una gestione più complessa, però garantiscono allo stesso tempo una più rapida modifica in caso di cambiamenti nelle funzionalità che vanno a gestire, rispetto alle API.

### 5.6.1 System Call Java

Le system call in ambiente Java sono implementate da due classi principali:

- **Runtime Class**;
- **ProcessBuilder Class**.

Entrambe le classi, anche se in modi differenti permettono di eseguire comandi in un processo separato. Per le funzionalità qui implementate si è scelta la classe ProcessBuilder, anche perché sviluppata più recentemente rispetto alla Runtime.

[1]API: Application Programming Interface – API per i comandi gLite

Come detto la classe ProcessBuilder permette di creare una system call attraverso la seguente istruzione:

*ProcessBuilder pb = new ProcessBuilder("myCommand", "myArg1", "myArg2")*

dove gli attributi myCommandi, myArg1, myArg2 rappresentano il comando da utilizzare e le sue relative opzioni. Una volta creato il processo attraverso il metodo ***start()*** della classe ProcessBuilder si ottiene una istanza della classe Process che serve a gestire le attività annesse al comando eseguito. Process fornisce i metodi seguenti:

- ***getInputStream()***: metodo che gestisce l'input stream del processo lanciato;
- ***getOutputStream()***: metodo che permette di gestire l'output stream del processo lanciato;
- ***getErrorStream()***: metodo che permette di gestire l'error stream del processo lanciato;
- ***waitFor()***: metodo che mette in attesa il thread corrente fino alla terminazione del processo lanciato;
- ***exitValue()***: metodo che ritorna il valore del processo, con zero terminato correttamente altrimenti errore;
- ***destroy()***: metodo che forza la terminazione anticipata del processo in esecuzione.

Da quanto sopra detto, è possibile gestire ogni system call lanciata, utilizzando in base alle esigenze, i vari metodi della classe Process. L'invocazione di un nuovo processo attraverso una system call può avvenire in due modi, o direttamente, invocando il comando (usando la shell di default), oppure indirettamente, tramite una nuova shell (es. "*/bin/bash -c command*" in sistemi Unix-like), che a sua volta esegue l'istruzione indicata. In questo secondo caso però occorre eventualmente ridefinire lo spazio ambiente in uso, cioè le variabili di sistema, che potrebbero essere utilizzate dal comando che si sta eseguendo. Vediamo adesso quali sono i comandi gLite (opportunamente implementati ognuno da una system call), che il modulo DriverGRID deve sfruttare per gestire le attività sulla griglia:

- **voms-proxy-init *–voms unina.it***: Creazione del certificato proxy per accesso al sito GRID-SCOPE;
- **voms-proxy-info**: Ritorna le informazioni sul certificato proxy, timeleft, path, etc...;
- **myproxy-init *-d -n –voms unina.it***: Creazione del myproxy;
- **myproxy-info**: Ritorna le informazioni sul certificato myproxy;
- **glite-wms-job-submit -a *pathname file jdl***: inoltra alla griglia il file jdl indicato;
- **glite-wms-job-status *ID_Job***: Restituisce le informazioni sullo stato del job indicato;
- **glite-wms-job-output *ID_Job***: Ritorna il file di output del job terminato con successo;
- **glite-wms-job-cancel -a *ID_Job***: Cancella dalla griglia il job identificato;
- l**cg-cp */grid/unina.it/<subpath>/fileoutput fileoutput_local***: Comando che permette di copiare il file di output dallo Storage Element alla UI.

A seguire alcune porzioni del codice implementato che descrivono delle system call.

Codice del metodo che permette la creazione di un certificato proxy:

```
//Metodo statico per la creazione del Proxy
private static void createProxy() throws DrExecException
{
 try
  {
   //Invocazione system call
   ProcessBuilder pb = new ProcessBuilder("/opt/glite"
            + "/bin/voms-proxy-init","-pwstdin","--voms","unina.it");

   //Lista variabili d'ambiente
   Map<String,String> env = pb.environment();

   //Inserimento variabili d'ambiente
   //per l'individuazione del certificato GRID
```

```
env.put("X509_USER_KEY", USERKEY);
env.put("X509_USER_CERT", USERCERT);

//Avvio processo
Process p = pb.start();

//Gestione dei tre standard input-output-error
OutputStream stdin = p.getOutputStream();
InputStream stdout = p.getInputStream();
InputStream stderror = p.getErrorStream();

//Lettura input
BufferedWriter buffer_input = new BufferedWriter
                              (new OutputStreamWriter(stdin));
PrintWriter input = new PrintWriter(buffer_input);
input.println(PASSPHRASE);

//Chiusura standard input
input.flush();
input.close();

//Variabili di lettura
String line = null;

//Creazione array per gestione standard output
ArrayList<String> array_buffer_output = new ArrayList<String>();

//Buffer standard output
BufferedReader buffer_output = new BufferedReader
                               (new InputStreamReader (stdout));
//Lettura output
while ((line = buffer_output.readLine()) != null)
    {
     System.out.println (Thread.currentThread() + "[Stdout] " + line);
     array_buffer_output.add(line);
    }

//Buffer standard error
BufferedReader buffer_error = new BufferedReader
                              (new InputStreamReader (stderror));
//Lettura error
while ((line = buffer_error.readLine ()) != null)
     System.out.println (Thread.currentThread() + "[Stderror] " + line);

//Chiusura standard output-error
buffer_output.close();
buffer_error.close();

//Attesa terminazione processo
p.waitFor();

if ( p.exitValue() != 0 )
  throw new DrExecException(Messages.getError(Messages.CREATE_PROXY));
else
  {
   //Creazione file di log per l'amministratore del sistema
   Log.createLogProxy(array_buffer_output);
   System.out.println(Thread.currentThread() + "Proxy Create!");
  }
}
//Catch per la system call
catch (IOException e)
{
 throw new DrExecException(Messages.getError(Messages.CREATE_PROXY));
}
```

```
  //Catch per il metodo waitFor
  catch (InterruptedException e)
   {
    throw new DrExecException(Messages.getError(Messages.CREATE_PROXY));
   }
 }
```

Codice del metodo che permette di inoltrare un job sulla griglia:

```
  //Metodo statico che permette di inoltrare job al sito GRID
  public static String submit(String experimentId,
                              String fileJDL) throws DrExecException
  {
   //Identificativo del job inviato
   String id_job = null;

   //Valore terminazione processo per gestione ciclo
   int exit = 0;
   //Valore di numero di volte che si puÃ² controllare il submit del job
   int max = 1;

   try
    {
     do {
      //Invocazione system call
      ProcessBuilder pb = new ProcessBuilder("/opt/glite"
                          + "/bin/glite-wms-job-submit","-a",fileJDL);

      //Avvio Processo
      Process p = pb.start();

      //Lettura standard output-error del processo lanciato
      InputStream stdout = p.getInputStream();
      InputStream stderror = p.getErrorStream();

      //Variabili di lettura
      String line = null;

      //Creazione array per gestione standard output
      ArrayList<String> array_buffer_output = new ArrayList<String>();

      //Buffer standar output
      BufferedReader buffer_output = new BufferedReader
                                     (new InputStreamReader (stdout));
      //Lettura output
      while ((line = buffer_output.readLine()) != null)
       {
        System.out.println (Thread.currentThread() + "[Stdout] " + line);
        array_buffer_output.add(line);
       }

      //Buffer standard error
      BufferedReader buffer_error = new BufferedReader
                                    (new InputStreamReader (stderror));
      //Lettura error
      while ((line = buffer_error.readLine ()) != null)
         System.out.println (Thread.currentThread() + "[Stderror] " + line);

      //Chiusura standard output-error
      buffer_output.close();
      buffer_error.close();

      //Attesa terminazione processo
      p.waitFor();
     //Valore terminazione processo
```

```
    exit = p.exitValue();

    if (exit != 0)
     {
      //Controllo numero di volte dell'accesso all'invio del job
      if (max == 0)
         throw new DrExecException(Messages.getError(Messages.SUBMIT_JOB));
      else
         {
          max = max - 1;
          //Distruzione processo
          p.destroy();
          //Attesa thread 300millisecondi
          Thread.sleep(300);
         }
     }
    else if (exit == 0)
     {
      //Valore id_job ottenuto
      id_job = array_buffer_output.get(PATHJOB);
      //Scrittura file di log per l'amministratore
      Log.createLogSubmit(array_buffer_output,experimentId);
     }

    System.out.println(Thread.currentThread() + "ID_JOB= " + id_job);
   }
   //Ciclo while di doppio controllo sul processo che
   //gestisce l'invio di un job
   while (exit != 0);

   //Scrittura eventuale del job_id nel database tramite experimentId

  }
  //Catch per la system call
  catch (IOException e)
   {
    throw new DrExecException(Messages.getError(Messages.SUBMIT_JOB));
   }
  //Catch per il metodo waitFor
  catch (InterruptedException e)
     {
      throw new DrExecException(Messages.getError(Messages.SUBMIT_JOB));
     }

  //Ritorna l'identificativo del job
  return id_job;
 }
```

## 5.7. Descrizione altre attività del DRMS

Il DRMS include anche altre funzionalità, oltre a quella di eseguire e cancellare esperimenti, come ad esempio la “traduzione”. Tale attività deve permettere di tradurre un file fornito in input dall'utente, a scelta tra quelli considerati ammissibili dall'infrastruttura (FITS, VOTABLE, ASCII e CSV), nel giusto formato richiesto dal modello scelto all'interno del componente DMM, e poi tradurre il file output nel formato iniziale fornito in input. Lo schema di traduzione è il seguente:

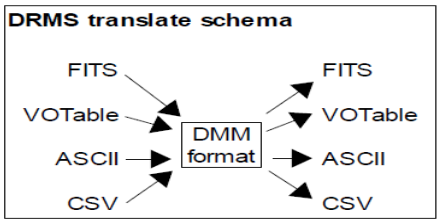



**Figura 30: Schema traduzione**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

Tale funzionalità è racchiusa nella classe ***Dataset***, che gestisce tutte le attività (che vengono utilizzate dal componente Framework), per la manipolazione dei dati. A livello architetturale è presente tra le librerie che compongono il componente DRMS, una classe denominata ***Dataset*** che ingloba vari metodi per la manipolazione dei dati che vengono di volta in volta forniti in input. La classe è la seguente:

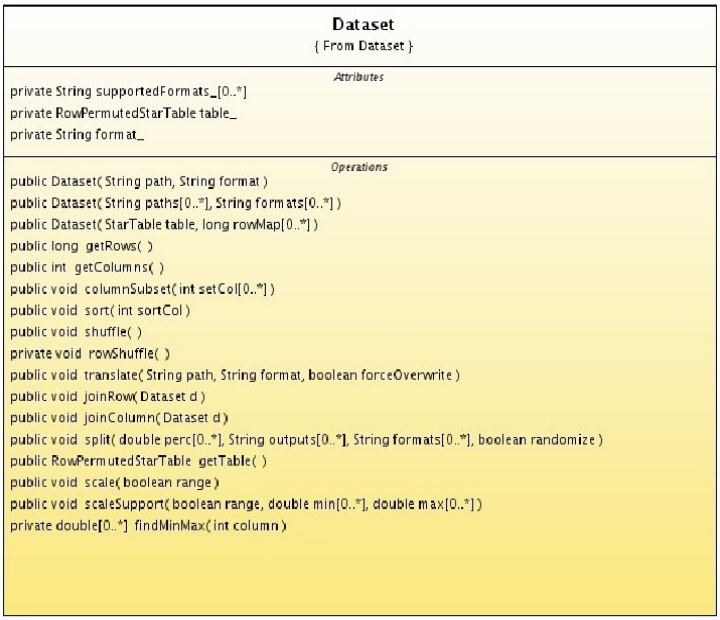



**Figura 31: Classe Dataset**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

Il DRMS include anche un'altra libreria, che ha il compito di fornire funzionalità di archiviazione su macchina stand alone e per l'infrastruttura GRID. In pratica sono presenti attività del tipo:

- ***Download***: metodo che permette di ottenere un file dal URI[1] fornito, e salvarlo nella cartella temporanea presente sulla macchina dove si trova la User Interface;
- ***Upload***: metodo che permette di trasferire uno specifico file dalla User Interface alla directory dell'utente presente sul File Store;
- ***Delete***: metodo che permette di cancellare uno specifico file o dalla User Interface e/o dal File Store;

[1]URI: Unified Resource Identifier

- ***Copy***: metodo che permette di copiare un file dalla directory di un utente verso un altro;
- ***getFile***: simile al metodo download, dove però ritorna come valore, un completo percorso URI, composto dalla combinazione, di un percorso specifico indicato, sommato al percorso parziale del file URI;
- ***executablePath***: metodo che ritorna il percorso completo di una directory dove risiedono tutti i file e librerie utili per l'esecuzione.

La classe (vedi figura 33), che implementa tali attività prende il nome di ***DrStorage***. Inoltre è presente una classe (vedi figura 33), denominata ***DrExecution***, che interagisce con il gestore del ***DriverSAManagement***, permettendo l'esecuzione di un esperimento su macchina stand alone.
Infine nella librerie del componente DRMS sono presenti due classi che gestiscono i vari errori d'esecuzione (***DrExecException***), e di archiviazione (***DrStorageException***).

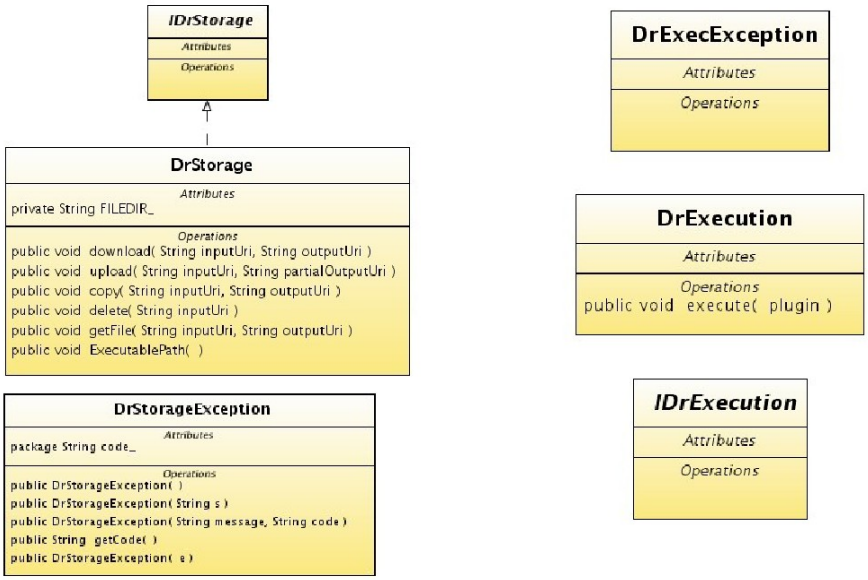



**Figura 32: Classe DrStorage, Classe DrExecution, Classe DrExecException**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

## 5.8. Gestione degli errori

La suite DAME prevede una gestione degli errori basta su una serie di codici d'errore che identificano il componente e il tipo di errore che si può presentare. La stringa d'errore è composta da 13 caratteri, così suddivisi:

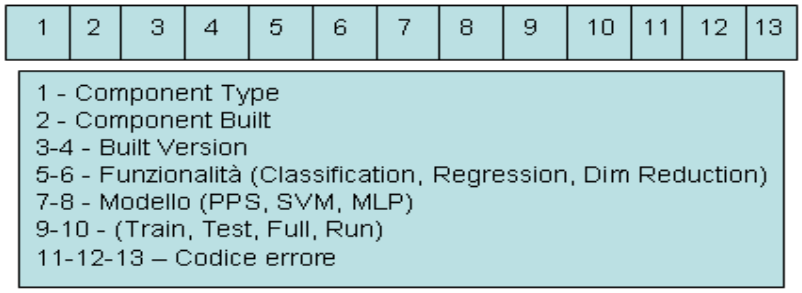



**Figura 33: Schema stringa errore**

**[RIF.] Sito web DAME: http://dame.dsf.unina.it/**

Ogni stringa di errore dispone di una notevole quantità di informazioni che possono essere utilizzate per individuare facilmente e correggere rapidamente l'errore. Nel caso del componente DRMS la stringa è: ***da01000000...,*** con gli ultimi tre che derivano dal tipo di errore che si può presentare nel componente.

## 5.9. Attività di Testing

L'attività di testing rappresenta il metodo attraverso il quale è possibile valutare l'efficienza del software implementato. In realtà lo scopo primario del testing (o collaudo), riguarda la capacità nell'individuare una serie di casi che possono generare una certa probabilità d'errore.
Di norma il software viene collaudato da due punti di vista differenti, si verifica la logica interna del codice attraverso la strategia “white-box”. Mentre i requisiti dell'applicazione vengono analizzati attraverso la strategia “black-box”.
Nel primo caso è possibile fare delle prove tese a verificare il funzionamento di determinate caratteristiche interne del codice (ad esempio cicli iterativi, flusso di dati, condizioni logiche).
Nella seconda strategia invece vengono realizzate delle prove adatte a verificare il funzionamento dell'applicazione senza conoscere la logica interna, ma solo analizzando il comportamento della funzionalità rispetto all'attività che deve svolgere. Per le applicazioni orientate agli oggetti, l'attività di testing, conviene eseguirla anche durante lo sviluppo, cioè testare singolarmente ogni classe (e di conseguenza le sue funzionalità), che si stanno implementando. Questo modo di ragionare però è fattibile solo se le classi rispecchiano una progettazione che si va a collocarsi nel concetto di alta coesione e basso accoppiamento (vedi paragrafo 5.2), perché si ottiene un codice fortemente modulare e quindi facilmente testabile nel singolo modulo.
Secondo quanto sopra detto, l'attività di testing sviluppata si inquadra nella logica della progettazione ad oggetti, quindi si è svolto un collaudo su ogni classe che si andava ad implementare, verificando sia la logica interna (white-box), dove necessario, sia la sola funzionalità (quindi black-box).
Ad esempio durante lo sviluppo della classe ***MyProxy*** sono stati svolti dei test per verificare le condizioni limite in cui si poteva trovare la funzione ***infoMyProxy***, come ad esempio l'assenza del certificato, la presenza di un certificato ancora valido e la rigenerazione di un certificato una volta scaduto.
Si sono anche effettuati con una serie di test sulla creazione del thread, che gestisce il codice che permette di analizzare ad intervalli di tempo definiti, lo stato di un job in esecuzione sulla griglia. Si sono in pratica testate le condizioni di terminazione del ciclo che gestisce il controllo dello stato del job. Questo perché, era fondamentale evitare qualsiasi situazione che potesse comportare un loop infinito.
Inoltre attraverso una classe opportunamente creata, si è simulato l'invio di più job sulla griglia, in modo da cercare di “stressare” l'applicazione, portandola ad eventuali errori. Tale situazione è stata molto utile soprattutto per verificare la consistenza dei metodi sincronizzati, in una situazione di multithreading.
In futuro, andranno comunque svolti dei test d'integrazione (una volta che DAMEGRID sarà totalmente integrato nella suite DAME), più completi rivolti a controllare le funzionalità e i comportamenti anomali che si possono verificare.

# Conclusione

Allo stato attuale, essendo DAMEGRID nella sua prima versione e non ancora integrato nella suite DAME, sarà sicuramente in futuro necessario modificare e/o aggiungere alcune funzionalità. Comunque vista l'architettura con cui si è sviluppato il tutto, ogni modifica e/o aggiunta non comporta grandi difficoltà.
Andrà poi sviluppata integralmente l'attività di scheduler, cioè la possibilità di scegliere, per ogni esperimento che viene lanciato, la piattaforma da utilizzare. Attività che verrà svolta attraverso una serie di parametri, che devono permette di selezionare la piattaforma migliore per quel particolare esperimento. Alcuni parametri possono ad esempio essere:

- Dimensione dei dati;
- Fasi di lavoro dell'esperimento (pre-processing, training, test, post-processing);
- Stato traffico code GRID;
- Disponibilità di worker node della GRID.

Allo stato attuale è stata solo definita la classe con un metodo ***getPlatform()***, che leggendo il valore da un file, permette di eseguire un esperimento in maniera alternata sulla piattaforma GRID e su piattaforma stand alone.

Concludo ringraziando il Dottore Massimo Brescia per avermi dato la possibilità di svolgere l'attività di tirocinio presso l'Osservatorio Astronomico di Capodimonte, e tutto lo staff DAME per essere sempre stati disponibili per qualsiasi aiuto e/o consiglio.
Inoltre ringrazio il Professore Adriano Peron per i consigli e suggerimenti profusi durante la stesura della presente tesi.

Un grazie a tutti

Giovanni.

# Bibliografia


- Magoules Frederic: Fundamentals of Grid Computing, Taylor francis, (2009).
- Roger S. Pressman: Principi di Ingegneria del software, McGraw-HIll, (2004).
- Ian Sommerville: Ingegneria del Software, Addison Wesley, (2005).
- Cay S. Horstmann, Gary Cornell: Core Java 2 volume 1 Fondamenti, Prentice Hall, (2005).
- Cay S. Horstmann, Gary Cornell: Core Java 2 volume 2 Funzioni avanzate, Prentice Hall, (2005).
- Mark G. Sobell: Practical Guide to Linux Commands, Editors, and Shell Programming, A (2nd Edition), Prentice Hall, (2009).
- Ercoli Luca: SysAdmin. Amministrazione di sistemi Linux, Boopen, 2010.

# Appendice A

**Tabella #2: Cockburn Seleziona Piattaforma**

| | | | |
|---|---|---|---|
| **USE CASE #1** | Seleziona Piattaforma | | |
| **Goal in Context** | Prevede la selezione della piattaforma(GRID-SA) su cui eseguire l'esperimento richiesto. La selezione avviene attraverso una stima su determinati parametri(scheduler) | | |
| **Scope & Level** | | | |
| **Preconditions** | Il componente FW esegue l'attività di lanciare un esperimento | | |
| **Success End Condition** | Si ottiene la piattaforma da utilizzare per l'esperimento da eseguire | | |
| **Filed End Condition** | Piattaforma da utilizzare non ottenuta | | |
| **Primary Actor** | DriverManager Agent | | |
| **Trigger** | Richiesta selezione della piattaforma da utilizzare | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Apre il file contenente i parametri per la selezione della piattaforma |
| | 3 | | Legge i parametri del file |
| | 4 | | Elabora i parametri per individuare la piattaforma da utilizzare |
| | 5 | | Fornisce la piattaforma su cui depistare l'esperimento |
| **Extension A:** Apertura del file non riuscito | **Step n°** | **Actor** | **System** |
| | 2a | | Errore: Apertura file non riuscita |
| **Extension B:** | **Step n°** | **Actor** | **System** |
| Elaborazione dei parametri non riuscita | 4a | | Errore: Elaborazione non eseguita |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #3: Cockburn Chiama DriverGRID**

| | | | |
|---|---|---|---|
| **USE CASE #2** | Chiama DriverGRID | | |
| **Goal in Context** | Chiamare il DriverGRID per inoltrare un job (esperimento) oppure per cancellare un job | | |
| **Scope & Level** | | | |
| **Preconditions** | La piattaforma GRID deve essere stata selezionata correttamente | | |
| **Success End Condition** | Chiamata DriverGRID eseguita | | |
| **Filed End Condition** | | | |
| **Primary Actor** | DriverManager Agent | | |
| **Trigger** | Richiesta chiamata DriverGRID | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Invoca il DriverGRID |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #4: Cockburn Chiama DriverSA**

| | | | |
|---|---|---|---|
| **USE CASE #3** | Chiama DriverSA | | |
| **Goal in Context** | Chiamare il DriverSA per effettuare le operazioni richieste in stand alone | | |
| **Scope & Level** | | | |
| **Preconditions** | La piattaforma  SA deve essere stata selezionata correttamente | | |
| **Success End Condition** | Chiamata DriverSA eseguita | | |
| **Filed End Condition** | | | |
| **Primary Actor** | DriverManager Agent | | |
| **Trigger** | Richiesta chiamata DriverSA | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Invoca il DriverSA |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #5: Cockburn Creazione Proxy/MyProxy**

| | | | |
|---|---|---|---|
| **USE CASE #4** | Creazione Proxy/Myproxy | | |
| **Goal in Context** | Creazione del certificato proxy e del myproxy | | |
| **Scope & Level** | | | |
| **Preconditions** | DRMS e certificato personale GRID presenti sulla macchina dove risiede la UI | | |
| **Success End Condition** | Certificato proxy e il myproxy ottenuti | | |
| **Filed End Condition** | Certificato proxy e/o il myproxy non ottenuti | | |
| **Primary Actor** | DriverGRID Agent | | |
| **Trigger** | Richiesta creazione proxy/myproxy | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Controlla proxy |
| | 3 | | Crea proxy |
| | 4 | | Controlla myproxy |
| | 5 | | Crea myproxy |
| | 6 | | Ritorna il certificato proxy e il myproxy |
| **Subvariation A:** | **Step n°** | **Actor** | **System** |
| Proxy già esistente e non ancora scaduto | 2a | | Non crea il proxy |
| | Ritorna allo step 4 | | |
| **Subvariation B:** | **Step n°** | **Actor** | **System** |
| Myproxy già esistente e non ancora scaduto | 4a | | Non crea il myproxy |
| | Ritorna allo step 6 | | |
| **Extension A:** | **Step n°** | **Actor** | **System** |
| Creazione proxy non avvenuta | 3a | | Errore: Certificato proxy non creato |
| **Extension B:** | **Step n°** | **Actor** | **System** |
| Certificato GRID personale(presente sulla macchina) scaduto | 2b | | Errore: Certificato personale GRID scaduto |
| **Extension C:** | **Step n°** | **Actor** | **System** |
| Controllo presenza myproxy non avvenuta | 4c | | Errore: Non posso controllare il myproxy |
| **Extension D:** | **Step n°** | **Actor** | **System** |
| Creazione myproxy non avvenuta correttamente | 5d | | Errore: Myproxy non creato |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #6: Cockburn Stato Job**

| | | | |
|---|---|---|---|
| **USE CASE #5** | Stato Job | | |
| **Goal in Context** | Controllare lo stato di un job precedentemente inviato alla griglia | | |
| **Scope & Level** | | | |
| **Preconditions** | UI installata correttamente sulla macchina dove risiede il DRMS e Myproxy ancora valido (*timeleft*) | | |
| **Success End Condition** | Stato job ottenuto | | |
| **Filed End Condition** | Stato job non ottenuto | | |
| **Primary Actor** | DriverGRID Agent | | |
| **Trigger** | Richiesta stato di un job/Aggiornamento stato di un job | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Richiesta stato di un job | |
| | 2 | | Recupera l'identificativo del job da controllare |
| | 3 | | Analizza lo stato del job |
| | 4 | | Ritorna in tempo reale lo stato del job |
| **Subvariation A:** | **Step n°** | **Actor** | **System** |
| Un sistema timer richiede ogni tot_tempo l'aggiornamento dello stato di un job | 1a | Aggiornamento stato di un job | |
| | Ritorna allo step 2 | | |
| **Extension A:** | **Step n°** | **Actor** | **System** |
| Il controllo dello stato del job non è avvenuto correttamente | 3a | | Errore: analisi stato job non eseguita |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #7: Cockburn Invio Job**

| | | | |
|---|---|---|---|
| **USE CASE #6** | Invio Job | | |
| **Goal in Context** | Inviare un job sulla griglia | | |
| **Scope & Level** | | | |
| **Preconditions** | UI installata correttamente sulla macchina dove risiede il DRMS e Myproxy ancora valido (*timeleft*) | | |
| **Success End Condition** | Invio Job eseguito correttamente | | |
| **Filed End Condition** | Invio di un job non avvenuto | | |
| **Primary Actor** | DriverGRID Agent | | |
| **Trigger** | Richiesta invio di un job | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Crea file jdl relativo all'esperimento da eseguire |
| | 3 | | Invia il file jdl(job) alla griglia |
| **Extension A:** | **Step n°** | **Actor** | **System** |
| Creazione file jdl non avvenuta correttamente | 2a | | Errore: creazione file jdl non riuscita |
| **Extension B:** | **Step n°** | **Actor** | **System** |
| Invio del job alla griglia non riuscito | 3b | | Errore: invio del job non riuscita |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #8: Cockburn Output Job**

| | | | |
|---|---|---|---|
| **USE CASE #7** | Output Job | | |
| **Goal in Context** | Ottenere l'output di un job terminato con successo sulla griglia | | |
| **Scope & Level** | | | |
| **Preconditions** | UI installata correttamente sulla macchina dove risiede il DRMS, Myproxy ancora valido (*timeleft*) e lo stato del job deve essere “Done (S*uccess*)” | | |
| **Success End Condition** | Output del job ottenuto | | |
| **Filed End Condition** | Output del job non ottenuto | | |
| **Primary Actor** | DriverGRID Agent | | |
| **Trigger** | Richiesta output di un job | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Recupera l'output e l'eventuale l'errore del job elaborato, direttamente o tramite lo Storage Element |
| | 3 | | Rende disponibili all'utente i file di output |
| **Extension A:** | **Step n°** | **Actor** | **System** |
| Recupero dei file del job terminato con successo non riuscita | 2a | | Errore: Impossibile recuperare i file |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #9: Cockburn Cancella Job**

| | | | |
|---|---|---|---|
| **USE CASE #8** | Cancella Job | | |
| **Goal in Context** | Permette di cancellare un job in esecuzione sulla griglia | | |
| **Scope & Level** | | | |
| **Preconditions** | UI installata correttamente sulla macchina dove risiede il DRMS e Myproxy ancora valido (*timeleft*) | | |
| **Success End Condition** | Cancellazione del job eseguita | | |
| **Filed End Condition** | Cancellazione del job non eseguita | | |
| **Primary Actor** | DriverGRID Agent | | |
| **Trigger** | Richiesta cancellazione di un job | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Recupera l'id del job da cancellare |
| | 3 | | Cancella il job in esecuzione sulla griglia |
| **Extensions A:** | **Step n°** | **Actor** | **System** |
| Cancellazione del job richiesto non avvenuta | 3a | | Errore: Cancellazione job non riuscita |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #10: Cockburn Lancio Esperimento**

| | | | |
|---|---|---|---|
| **USE CASE #9** | Lancio Esperimento | | |
| **Goal in Context** | Invocazione DriverManager per effettuare la scelta della piattaforma da utilizzare per eseguire l'esperimento | | |
| **Scope & Level** | | | |
| **Preconditions** | | | |
| **Success End Condition** | DriverManager attivato | | |
| **Filed End Condition** | | | |
| **Primary Actor** | Framework Component | | |
| **Trigger** | Richiesta lancio esperimento | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Invia le informazioni dell'esperimento selezionato al DriverManager |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #11: Cockburn Cancellazione esperimento su GRID**

| | | | |
|---|---|---|---|
| **USE CASE #10** | Cancellazione esperimento su GRID | | |
| **Goal in Context** | Invocazione DriverManager per poter cancellare dalla griglia l'esperimento selezionato | | |
| **Scope & Level** | | | |
| **Preconditions** | | | |
| **Success End Condition** | DriverManager attivato | | |
| **Filed End Condition** | | | |
| **Primary Actor** | Framework Component | | |
| **Trigger** | Richiesta cancellazione dalla griglia di un esperimento(job) | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Invia al DriverManager le informazioni dell'esperimento da cancellare sulla griglia |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

**Tabella #12: Cockburn Esecuzione esperimento**

| | | | |
|---|---|---|---|
| **USE CASE #11** | Esecuzione esperimento | | |
| **Goal in Context** | Esegue l'esperimento scelto sulla macchina stand alone | | |
| **Scope & Level** | | | |
| **Preconditions** | La piattaforma SA deve essere stata selezionata correttamente | | |
| **Success End Condition** | Esecuzione esperimento eseguito correttamente | | |
| **Filed End Condition** | Esecuzione esperimento non riuscito | | |
| **Primary Actor** | DriverSA Agent | | |
| **Trigger** | Richiesta esecuzione esperimento | | |
| **Description** | **Step n°** | **Actor** | **System** |
| | 1 | Trigger | |
| | 2 | | Controlla presenza di tutti i file utili per l'esperimento |
| | 3 | | Esecuzione esperimento |
| **Extension A:** | **Step n°** | **Actor** | **System** |
| I file utli per l'esperimento non sono tutti presenti | 2a | | Errore: non tutti i file sono presenti |
| **Extension B:** | **Step n°** | **Actor** | **System** |
| Elaborazione dell'esperimento richiesto non riuscita | 3b | | Errore: esecuzione esperimento non riuscita |
| **Open Iusses** | | | |
| **Superordinates** | | | |
| **Subordinates** | | | |

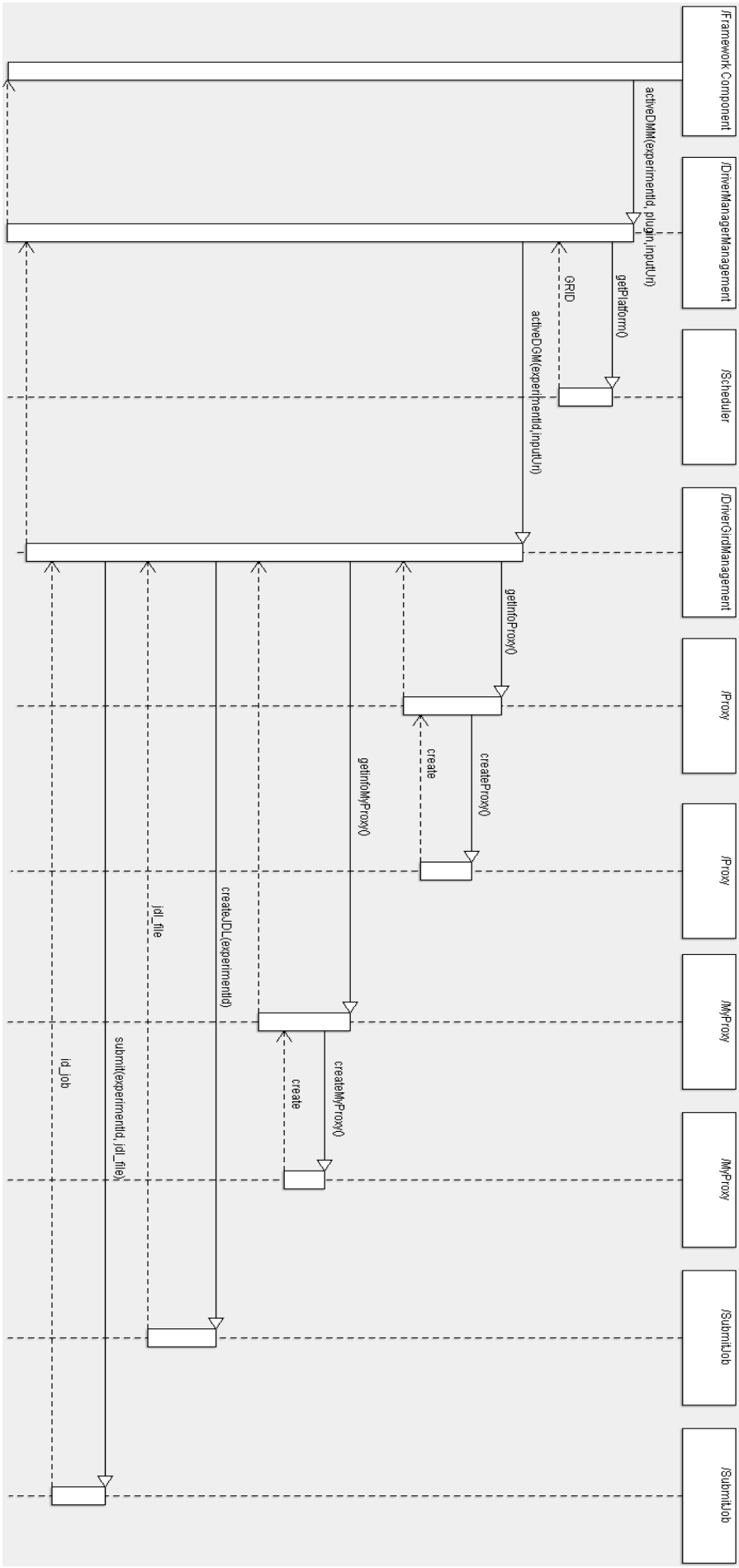



**Figura 34: SD Lancio esperimento su GRID con creazione Proxy e MyProxy**

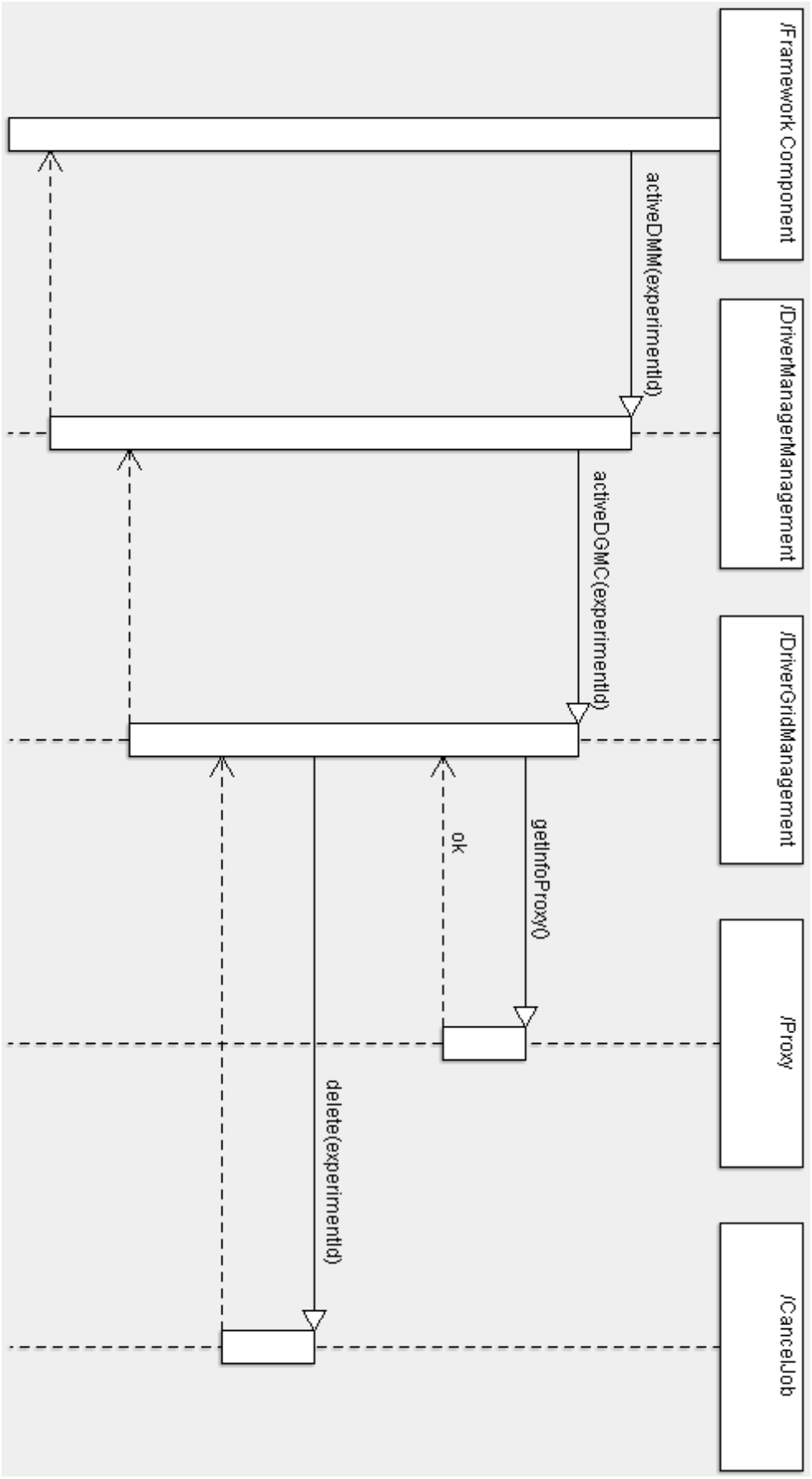



**Figura 35: SD Cancellazione job**

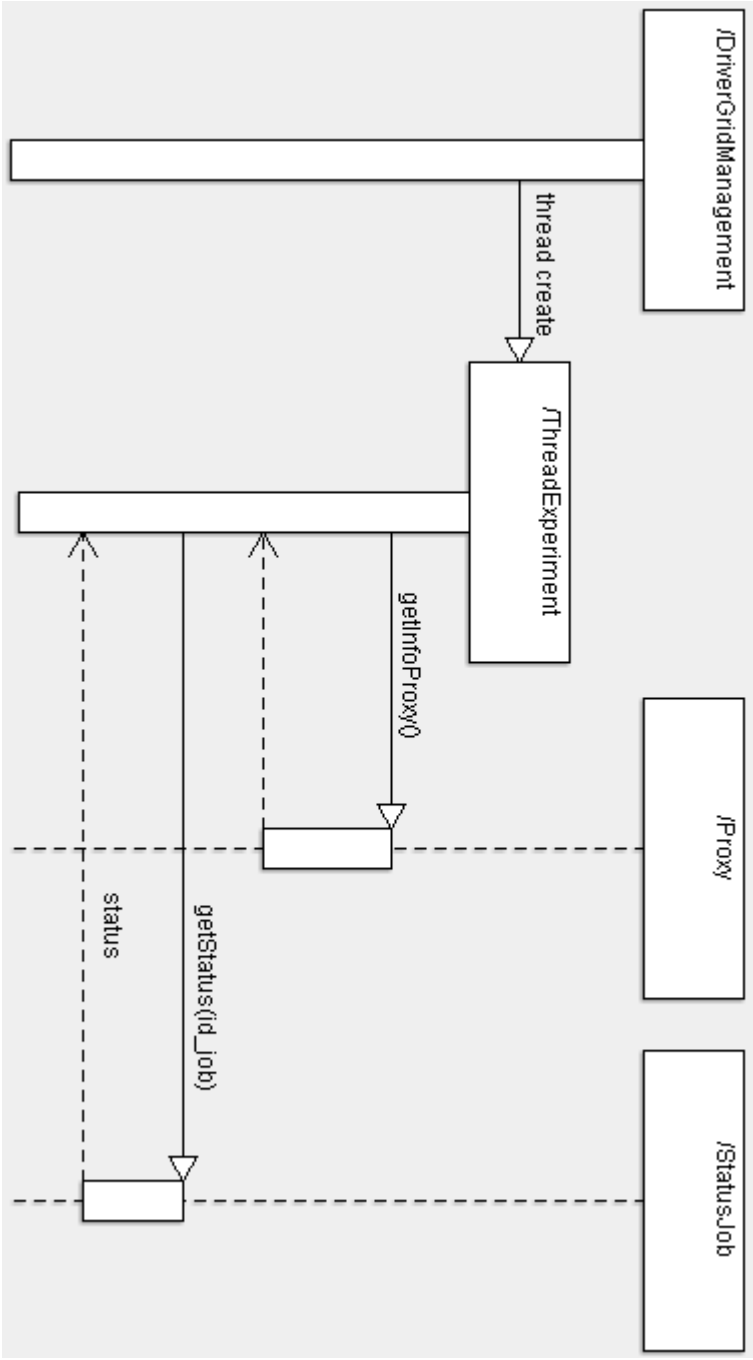



**Figura 36: SD Controllo status job**

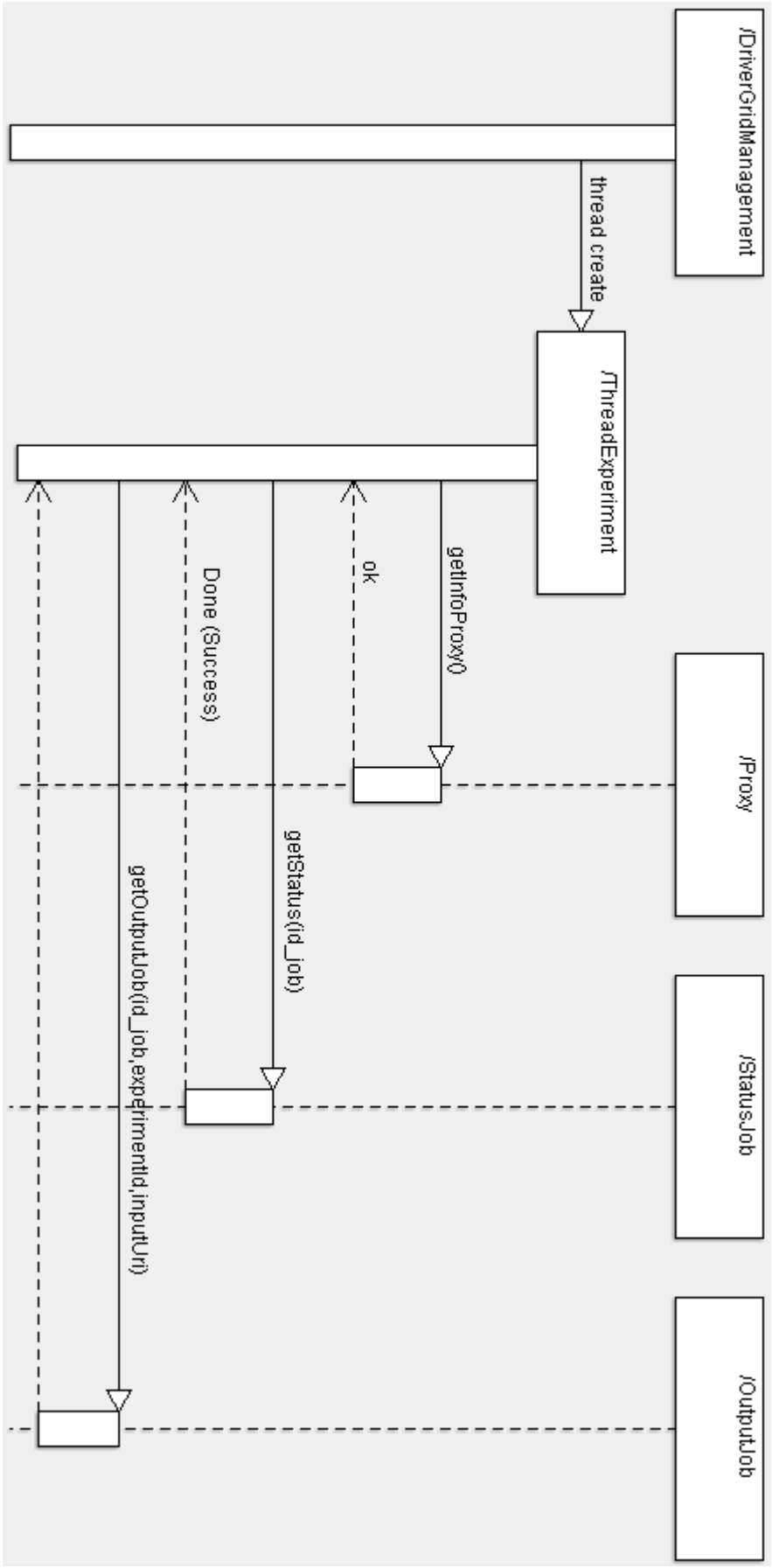



**Figura 37: SD Gestione output job terminato con successo**

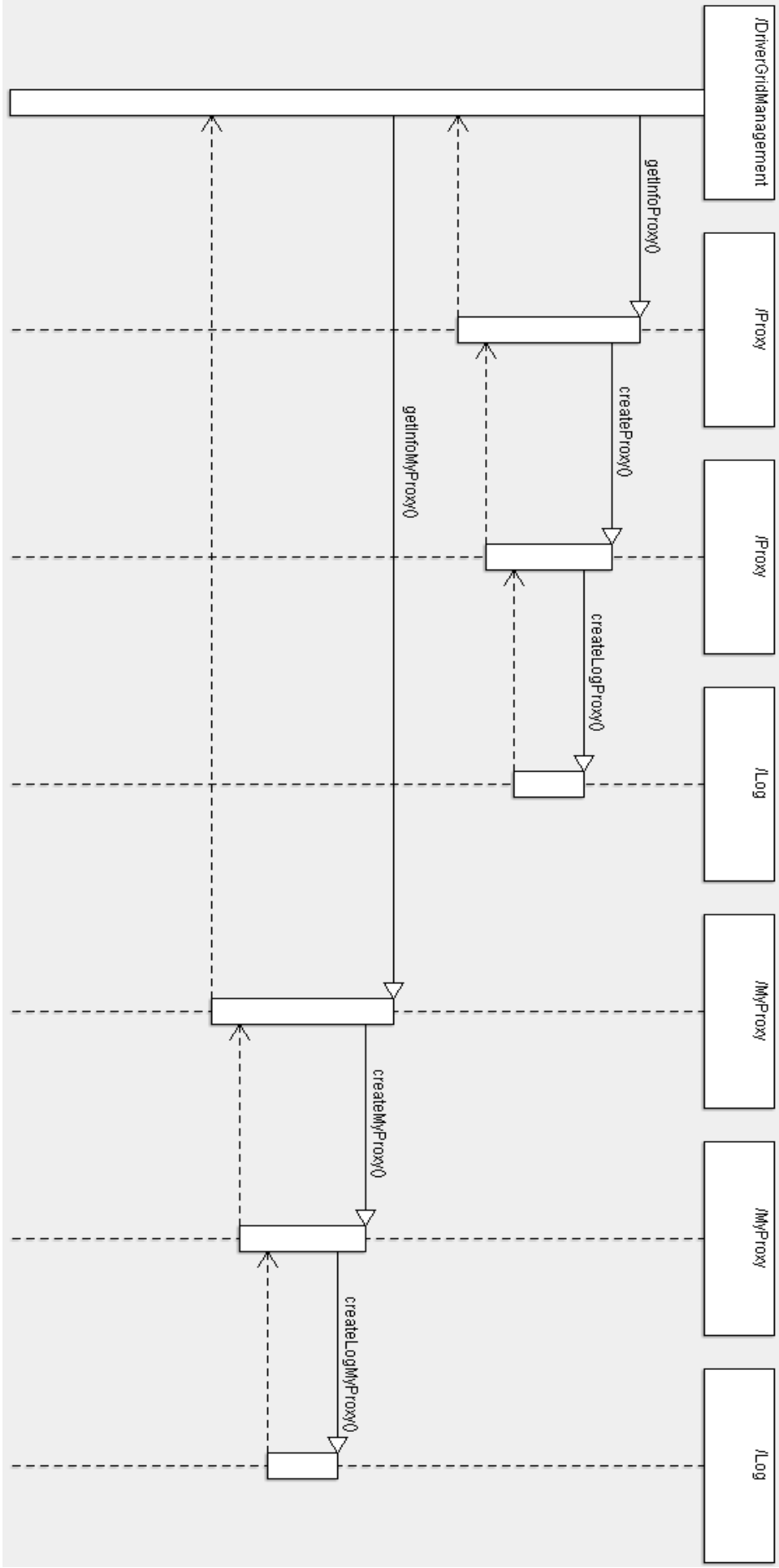



**Figura 38: SD Creazione Proxy e MyProxy con generazione file di Log**

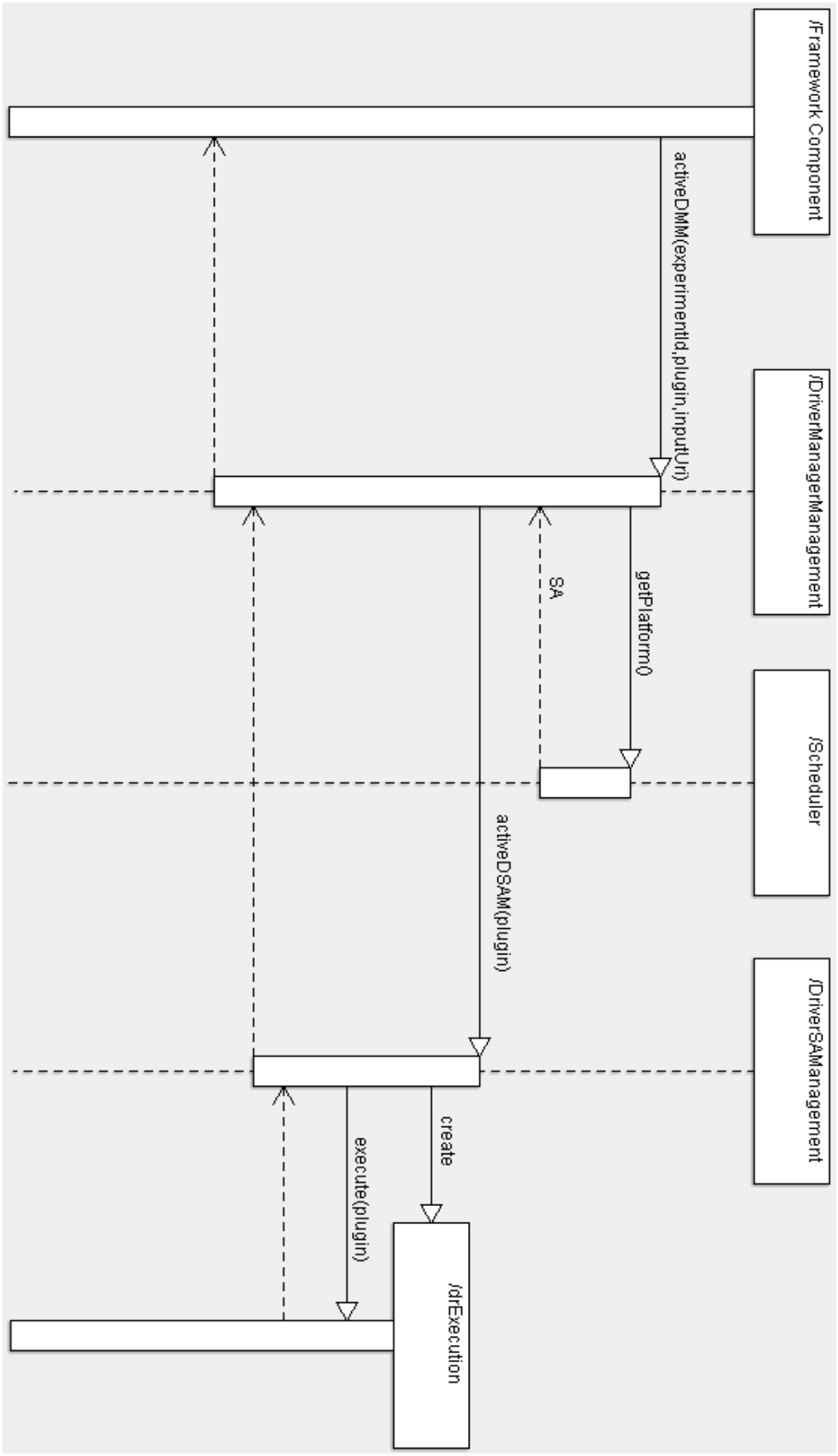



**Figura 39: SD Lancio esperimento su macchina SA**